This book sets out to demonstrate the power of the Markov random field (MRF) in vision. It treats the MRF both as a tool for modeling image data and, coupled with a set of recently developed algorithms, as a means of making inferences about images. The inferences concern underlying image and scene structure to solve problems such as image reconstruction, image segmentation, 3D vision, and object labeling. This chapter is designed to present some of the main concepts used in MRFs, both as a taster and as a gateway to the more detailed chapters that follow, as well as a stand-alone introduction to MRFs.

The unifying ideas in using MRFs for vision are the following:

- Images are dissected into an assembly of nodes that may correspond to pixels or agglomerations of pixels.
- Hidden variables associated with the nodes are introduced into a model designed to “explain” the values (colors) of all the pixels.
- A joint probabilistic model is built over the pixel values and the hidden variables.
- The direct statistical dependencies between hidden variables are expressed by explicitly grouping hidden variables; these groups are often pairs depicted as edges in a graph.

These properties of MRFs are illustrated in figure 1.1. The graphs corresponding to such MRF problems are predominantly gridlike, but may also be irregular, as in figure 1.1(c). Exactly how graph connectivity is interpreted in terms of probabilistic conditional dependency is discussed a little later.

The notation for image graphs is that the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of vertices $\mathcal{V} = \{1, 2, \ldots, i, \ldots, N\}$ corresponding, for example, to the pixels of the image, and a set of edges $\mathcal{E}$ where a typical edge is $(i, j)$, $i, j \in \mathcal{V}$, and edges are considered to be undirected, so that $(i, j)$ and $(j, i)$ refer to the same edge. In the superpixel graph of figure 1.1), the nodes are superpixels, and a pair of superpixels forms an edge in $\mathcal{E}$ if the two superpixels share a common boundary.

The motivation for constructing such a graph is to connect the hidden variables associated with the nodes. For example, for the task of segmenting an image into foreground and background, each node $i$ (pixel or superpixel) has an associated random variable $X_i$ that
Introduction to Markov Random Fields

Figure 1.1
Graphs for Markov models in vision. (a) Simple 4-connected grid of image pixels. (b) Grids with greater connectivity can be useful—for example, to achieve better geometrical detail (see discussion later)—as here with the 8-connected pixel grid. (c) Irregular grids are also useful. Here a more compact graph is constructed in which the nodes are superpixels—clusters of adjacent pixels with similar colors.

may take the value 0 or 1, corresponding to foreground or background, respectively. In order to represent the tendency of matter to be coherent, neighboring sites are likely to have the same label. So where \((i, j) \in \mathcal{E}\), some kind of probabilistic bias needs to be associated with the edge \((i, j)\) such that \(X_i\) and \(X_j\) tend to have the same label—both 0 or both 1. In fact, any pixels that are nearby, not merely adjacent, are likely to have the same label. On the other hand, explicitly linking all the pixels in a typical image, whose foreground/background labels have correlations, would lead to a densely connected graph. That in turn would result in computationally expensive algorithms. Markov models explicitly represent only the associations between relatively few pairs of pixels—those pixels that are defined as neighbors because of sharing an edge in \(\mathcal{E}\). The great attraction of Markov Models is that they leverage a knock-on effect—that explicit short-range linkages give rise to implied long-range correlations. Thus correlations over long ranges, on the order of the diameters of typical objects, can be obtained without undue computational cost. The goal of this chapter is to investigate probabilistic models that exploit this powerful Markov property.

1.1 Markov Chains: The Simplest Markov Models

In a Markov chain a sequence of random variables \(X = (X_1, X_2, \ldots)\) has a joint distribution specified by the conditionals \(P(X_i \mid X_{i-1}, X_{i-2}, \ldots, X_1)\). The classic tutorial example [381, sec. 6.2] is the weather, so that \(X_i \in \mathcal{L} = \{\text{sunny, rainy}\}\). The weather on day \(i\) can be influenced by the weather many days previous, but in the simplest form of Markov chain, the dependence of today’s weather is linked explicitly only to yesterday’s weather. It is also linked implicitly, as a knock-on effect, to all previous days. This is a first-order Markov assumption, that

\[
P(X_i \mid X_{i-1}, X_{i-2}, \ldots, X_1) = P(X_i \mid X_{i-1}).
\]

(1.1)

This is illustrated in figure 1.2. The set of conditional probabilities \(P(X_i \mid X_{i-1})\) is in fact a \(2 \times 2\) matrix. For example:
An interesting and commonly used special case is the stationary Markov chain, in which the matrix

\[ M_i(x, x') = P(X_i = x \mid X_{i-1} = x') \]  

(1.2)

is independent of time \( i \), so that \( M_i(\cdot, \cdot) = M_{i-1}(\cdot, \cdot) \). In the weather example this corresponds to the assumption that the statistical dependency of weather is a fixed relationship, the same on any day.

We will not dwell on the simple example of the Markov chain, but a few comments may be useful. First, the first-order explicit structure implicitly carries longer-range dependencies, too. For instance, the conditional dependency across three successive days is obtained by multiplying together the matrices for two successive pairs of days:

\[ P(X_i = x \mid X_{i-2} = x'') = \sum_{x' \in \mathcal{L}} M_i(x, x')M_{i-1}(x', x''). \]  

(1.3)
Thus the Markov chain shares the elegance of Markov models generally, which will recur later with models for images, that long-range dependencies can be captured for the “price” of explicitly representing just the immediate dependencies between neighbors. Second, higher-order Markov chains, where the explicit dependencies go back farther than immediate neighbors, can also be useful. A famous example is “predictive text,” in which probable letters in a typical word are characterized in terms of the two preceding letters—taking just the one preceding letter does not give enough practical predictive power. Predictive text, then, is a second-order Markov chain.

The directed graph in figure 1.2a) is a graphical representation of the fact that, for a Markov chain, the joint density can be decomposed as a product of conditional densities:

\[ P(x) = P(x_N | x_{N-1}) \ldots P(x_i | x_{i-1}) \ldots P(x_2 | x_1) P(x_1), \quad (1.4) \]

where for simplicity, in a popular abuse of notation, \( P(x) \) denotes \( P(X = x) \) and, similarly, \( P(x_i | x_{i-1}) \) denotes \( P(X_i = x_i | X_{i-1} = x_{i-1}) \). This convention is used frequently throughout the book. An alternative formalism that is commonly used is the undirected graphical model. Markov chains can also be represented in this way (figure 1.2c), corresponding to a factorized decomposition:

\[ P(x) = \Phi_{N,N-1}(x_N, x_{N-1}) \ldots \Phi_{i,i-1}(x_i, x_{i-1}) \ldots \Phi_{2,1}(x_2, x_1), \quad (1.5) \]

where \( \Phi_{i,i-1} \) is a factor of the joint density. It is easy to see, in this simple case of the Markov chain, how the directed form (1.4) can be reexpressed in the undirected form (1.5). However, it is not the case in general, and in particular in 2D images, that models expressed in one form can easily be expressed in the other. Many of the probabilistic models used in computer vision are most naturally expressed using the undirected formalism, so it is the undirected graphical models that dominate in this book. For details on directed graphical models see [216, 46].

**1.2 The Hidden Markov Model (HMM)**

Markov models are particularly useful as prior models for state variables \( X_i \) that are to be inferred from a corresponding set of measurements or observations \( z = (z_1, z_2, \ldots, z_i, \ldots, z_N) \). The observations \( z \) are themselves considered to be instantiations of a random variable \( Z \) representing the full space of observations that can arise. This is the classical situation in speech analysis [381, sec. 6.2], where \( z_i \) represents the spectral content of a fragment of an audio signal, and \( X_i \) represents a state in the time course of a particular word or phoneme. It leads naturally to an inference problem in which the posterior distribution for the possible states \( X \), given the observations \( z \), is computed via Bayes’s formula as

\[ P(X = x | Z = z) \propto P(Z = z | X = x) P(X = x). \quad (1.6) \]

Here \( P(X = x) \) is the prior distribution over states—that is, what is known about states \( X \) in the absence of any observations. As before, (1.6) is abbreviated, for convenience, to
The Hidden Markov Model (HMM)

\[ P(\mathbf{x} | \mathbf{z}) \propto P(\mathbf{z} | \mathbf{x}) P(\mathbf{x}). \]  

(1.7)

The omitted constant of proportionality would be fixed to ensure that \( \sum_{\mathbf{x}} P(\mathbf{x} | \mathbf{z}) = 1 \). Often multiple models are considered simultaneously, and in that case this is denoted

\[ P(\mathbf{x} | \mathbf{z}, \omega) \propto P(\mathbf{z} | \mathbf{x}, \omega) P(\mathbf{x} | \omega), \]  

(1.8)

where the model parameters \( \omega \in \Omega \) may determine the prior model or the observation model or both. The constant of proportionality in this relation would of course depend on \( \mathbf{z} \) and on \( \omega \).

The prior of an HMM is itself represented as a Markov chain, which in the first-order case was decomposed as a product of conditional distributions (1.4). The term \( P(\mathbf{z} | \mathbf{x}) \) is the likelihood of the observations, which is essentially a measure of the quality of the measurements. The more precise and unambiguous the measuring instrument, the more the likelihood will be compressed into a single, narrow peak. This captures the fact that a more precise instrument produces more consistent responses \( \mathbf{z} \), under a given condition represented by the state \( \mathbf{X} = \mathbf{x} \). It is often assumed—and this is true of the models used in many of the chapters of this book—that observations are independent across sites. The observation at site \( i \) depends only on the corresponding state. In other words:

\[ P(\mathbf{z} | \mathbf{x}) = P(z_N | x_N) P(z_{N-1} | x_{N-1}) \ldots P(z_1 | x_1). \]  

(1.9)

The directed graphical model for the conditional dependencies of such a first-order HMM is given in figure 1.3a). The figure captures the conditional dependencies both of the underlying

![Directed Graphical Model](image)

Figure 1.3
A first-order hidden Markov model (HMM). (a) A directed graph is used to represent the dependencies of a first-order HMM, with its Markov chain prior, and a set of independently uncertain observations. (b) Alternatively the HMM can be represented as an undirected graphical model (see text).
Markov chain and of the independence of the observations. Alternatively, an HMM can be expressed as an undirected graphical model, as depicted in figure 1.3(b), in which the prior is decomposed as in (1.5), and the likelihood is

\[ P(z \mid x) = \Phi_N(x_N) \Phi_{N-1}(x_{N-1}) \cdots \Phi_1(x_1), \]

where trivially \( \Phi_i(x_i) = P(z_i \mid x_i) \).

Discrete HMMs, with a finite label set \( \mathcal{L} \), are largely tractable. Rabiner and Juang [382] set out three canonical problems for HMMs, and algorithms to solve them. The problems are the following:

1. **Evaluating the observation probability** \( P(z \mid \omega) \)  
   In this problem there is no explicit state dependence, because it has been “marginalized” out by summation over states:

\[ P(z \mid \omega) = \sum_{x \in \mathcal{L}^N} P(z \mid x, \omega) P(x \mid \omega). \]

The main application of this evaluation is to determine which of a set of known models fits the data best:

\[ \max_{\omega \in \Omega} P(z \mid \omega). \]

The quantity \( P(z \mid \omega) \) is also known as the *evidence* [328] for the model \( \omega \) from the data \( z \).

2. **MAP estimation**  
   Given a model \( \omega \) and a set of data \( z \), estimate the most probable (maximum a posteriori) sequence \( x \) of states as the mode of the posterior distribution (1.8).

3. **Parameter estimation**  
   Given a set of data \( z \), estimate the parameters \( \omega \in \Omega \), a continuous parameter space that best fits the data. This is the problem that must be solved to build a particular model from training data. It is closely related to the model selection problem above, in that both maximize \( P(z \mid \omega) \), the difference being that the model space \( \Omega \) is, respectively, discrete or continuous.

These three problems are essentially solved by using two algorithms and variants of them.

The first problem requires the *forward* algorithm that computes a marginal distribution node \( i \) from the distribution at the previous node \( i - 1 \):

\[ P(x_i, z_1, \ldots, z_i \mid \omega) = P(z_i \mid x_i, \omega) \sum_{x_{i-1}} P(x_i \mid x_{i-1}, \omega) P(x_{i-1}, z_1, \ldots, z_{i-1} \mid \omega). \]

This is a special case of *Belief Propagation* (BP) that will be discussed later in this chapter and in various subsequent chapters in the book. In fact there are two forms of BP [367, 46], and this one is an example of *sum-product* BP. (The name derives from the summation and product steps in (1.13).) The other form is described shortly. In the case of the HMM, where the underlying prior model is simply a Markov chain, sum-product belief propagation is
The Hidden Markov Model (HMM) is quite straightforward and is an exact algorithm for computing the marginal posteriors. After one complete forward pass, the final marginal distribution is $P(x_N, z | \omega)$, and so finally

$$P(z | \omega) = \sum_{x_N} P(x_N, z | \omega) \quad (1.14)$$

can be computed as the evidence for a known model $\omega$ that solves problem 1 above. The forward pass (1.13) constitutes half of BP, the remaining part being a backward pass that recurs from node $N$ back to node 1 (details omitted here, but see [382]). Using the forward and backward passes together, the full set of marginal posterior distributions

$$P(x_i | z, \omega), \ i = 1, \ldots, N \quad (1.15)$$

can be computed. This is required for problem 3 above, in order to compute the expected values of the sufficient statistics that are needed to estimate the parameters $\omega$ by expectation maximization [121]—also known in the speech analysis literature as the Baum–Welch method [381].

The second algorithm is the Viterbi algorithm, a dynamic programming optimization algorithm applied to the state sequence $x$. It is also equivalent to a special case of max-product belief propagation, which also is mentioned quite frequently in the book. The aim is to solve the second problem above, computing the MAP estimate of the state vector as

$$\hat{x} = \arg \max_x P(x | z, \omega) \quad (1.16)$$

via a forward recursion:

$$P_i(x_i) = P(z_i | x_i, \omega) \max_{x_{i-1}} P(x_i | x_{i-1}, \omega) P_{i-1}(x_{i-1}) \quad (1.17)$$

where $P_i$ is defined by

$$P_i(x_i) \equiv \max_{x_1, \ldots, x_{i-1}} \ P(x_1, \ldots, x_i, z_1, \ldots, z_i | \omega). \quad (1.18)$$

Each forward step of the recursion can be viewed as a message-passing operation. In step $i - 1$ of the computation, node $i - 1$ sends the message $P_i(x_i)$ to node $i$.

After the forward steps are complete, the final component $\hat{x}_N$ of the MAP solution $\hat{x}$ can be computed as

$$\hat{x}_N = \arg \max_{x_N} P_n(x_N). \quad (1.19)$$

This is followed by a backward recursion

$$\hat{x}_{i-1} = \arg \max_{x_{i-1}} P(\hat{x}_i | x_{i-1}, \omega) P_{i-1}(x_{i-1}), \quad (1.20)$$

after which all components of $\hat{x}$ are determined.
The purpose of the discussion in this section has been largely to explain the nature of hidden variables in simple Markov models, as a precursor to later discussion of hidden variables in the more complex, two-dimensional kinds of models that are used in vision. However, even in vision the discrete HMM structure has some direct applications. It has proved useful for representing temporal problems that are somewhat analogous to speech analysis, but in which the audio input is replaced by a time sequence of visual features. Well-known examples include the recognition of American Sign Language [449] and the recognition of hand gestures for command and control [522]. This book deals mainly with discrete Markov models—that is, ones in which the states of each \( X_i \) belong to a finite set \( \mathcal{L} \).

However, in vision by far the greater application of timelike HMMs employs continuous state-space to represent position, attitude, and shape in visual tracking [333, 477, 50, 253, 209, 124]. In such continuous settings the HMM becomes a classical or nonclassical form of Kalman filter. Both exact solutions to the estimation problems that arise, and efficient approximate solutions, are much studied, but are outside the scope of this book.

### 1.3 Markov Models on Trees

In the following section 1.4, Markov Random Fields (MRFs) are defined as probabilistic models over undirected graphs. On the way there, we now consider undirected models on trees as intermediate in complexity between the linear graphs—chains and HMMs—of section 1.2, and graphs of unrestricted connectivity. Clearly the HMM graph (figure 1.3b) is a special case of an undirected model on a tree. Trees appear to be of intermediate complexity but, perhaps surprisingly, turn out to be closer to HMMs, in that inference can be performed exactly. The Viterbi and forward-backward algorithms for HMMs generalize to two different kinds of message passing on trees. However, once the nodes on two leaves of a tree are coalesced into a single leaf—for example, leaves \( \text{b} \) and \( \text{d} \) in figure 1.4—a circuit may be formed in the resulting graph, and message-passing algorithms are no longer an exact solution to the problems of inference.

As with Markov chains and HMMs, in undirected trees the topological structure conveys two aspects of the underlying probabilistic model. First are the conditional independence properties, that:

\[
P(x_i | \{x_j, j \neq i\}) = P(x_i | \{x_j, (i, j) \in \mathcal{E}\}).
\]

The set \( B_i = \{ j : (i, j) \in \mathcal{E} \} \) is known as the Markov blanket of \( i \), its neighbors in the tree (or generally graph) \( \mathcal{G} \). The second aspect is the decomposition of the joint distribution, the generalization of (1.5) and (1.10). How can a distribution with this independence property be constructed? The answer is, as a distribution that is factorized over the edges of the tree:

\[
P(x) = \prod_{(i,j) \in \mathcal{E}} F_{i,j}(x_i, x_j).
\]
1.3 Markov Models on Trees

1.3.1 Inference on Trees: Belief Propagation (BP)

The message-passing formulation of the Viterbi algorithm can be generalized to find marginal distributions over individual variables and the MAP estimate in a tree structured graphical model. The resulting algorithm is known as belief propagation \[367\] and has two variants: max-product, for computing the MAP solution, and sum-product (mentioned earlier), which allows computation of marginals of individual random variables.

Max-product message passing is similar in spirit to dynamic programming algorithms. Like the Viterbi algorithm, it works by passing messages between tree nodes in two stages. In the first stage, messages are passed from the leaf nodes to their parents, which in turn pass messages to their parents, and so on until the messages reach the root node. The message $m_{i \rightarrow j}(x_j)$ from a node $i$ to its parent $j$ is computed as

$$m_{i \rightarrow j}(x_j) = \max_{x_i} P(x_j, x_i) \prod_{k \in \mathcal{N}_c(i)} m_{k \rightarrow i}(x_i) \quad (1.23)$$

where $\mathcal{N}_c(i)$ is the set of all children of node $i$. The MAP label of the variable at the root $r$ of the tree can be computed as

$$\hat{x}_r = \arg \max_{x_r} \prod_{k \in \mathcal{N}_c(r)} m_{k \rightarrow r}(x_r). \quad (1.24)$$

Given the MAP label $\hat{x}_p$ of a variable $X_p$, the label of any of its children $i$ can be found as

$$\hat{x}_i = \max_{x_i} P(\hat{x}_p, x_i) \prod_{k \in \mathcal{N}_c(i)} m_{k \rightarrow i}(x_i). \quad (1.25)$$
1.3.2 Example: Max-Product BP on a Five-Node Model

Consider the undirected, tree-structured graphical model shown in figure 1.4. The joint distribution factorizes as

\[ P(\mathbf{x}) = P(x_a, x_b)P(x_a, x_c)P(x_c, x_e)P(x_c, x_d). \]  

(1.26)

The messages computed by max-product are

\[
\begin{align*}
    m_{d \rightarrow c}(x_c) &= \max_{x_d} P(x_c, x_d) \\
    m_{e \rightarrow c}(x_c) &= \max_{x_e} P(x_c, x_e) \\
    m_{c \rightarrow a}(x_a) &= \max_{x_c} P(x_a, x_c)m_{e \rightarrow c}(x_c)m_{d \rightarrow c}(x_c) \\
    m_{b \rightarrow a}(x_a) &= \max_{x_b} P(x_a, x_b).
\end{align*}
\]

(1.27)\(\ldots\) (1.30)

The MAP labels can be found as

\[
\begin{align*}
    \hat{x}_a &= \max_{x_a} m_{b \rightarrow a}(x_a)m_{c \rightarrow a}(x_a) \\
    \hat{x}_b &= \max_{x_b} P(\hat{x}_a, x_b) \\
    \hat{x}_c &= \max_{x_c} P(\hat{x}_a, x_c)m_{e \rightarrow c}(x_c)m_{d \rightarrow c}(x_c) \\
    \hat{x}_d &= \max_{x_d} P(\hat{x}_c, x_d) \\
    \hat{x}_e &= \max_{x_e} P(\hat{x}_c, x_e).
\end{align*}
\]

(1.31)\(\ldots\) (1.35)

The sum product BP algorithm computes the marginal distributions \(P(x_i)\) for all variables \(X_i\). It essentially works in a way similar to max-product BP (1.23), except that rather than taking the max, a sum is performed over the different labels:

\[
\begin{align*}
    m_{i \rightarrow j}(x_j) &= \sum_{x_i} P(x_j, x_i) \prod_{k \in \mathcal{N}_c(i)} m_{k \rightarrow i}(x_i) \\
    m_{k \rightarrow i}(x_i) &= \sum_{x_k} P(x_i, x_k) \prod_{j \in \mathcal{N}_c(i) \setminus \{i\}} m_{j \rightarrow i}(x_j)
\end{align*}
\]

(1.36)

where \(\mathcal{N}_c(i)\) is the set of all children of node \(i\). The marginal \(P(x_i)\) can be computed by taking the product of the messages sent to the root node \(i\):

\[
P(x_i) = \prod_{k \in \mathcal{N}_c(i)} m_{k \rightarrow i}(x_i).
\]

(1.37)

Now, by successively rearranging the tree so that each node \(i\) in turn becomes the root node, \(P(x_i)\) can be computed for all nodes \(i\).
1.4 MRFs: Markov Models on Graphs

At the start of the chapter, the choice of graphs for image processing was motivated by the need to establish dependencies between pixels that are nearby in an image. The graphs that were proposed for that purpose are not trees, but contain cycles of edges, typically many cycles, as in figure 1.1. The representation of independence in undirected graphs follows the methodology given above for trees (1.21). The random variable at a node is dependent directly only on random variables in its Markov blanket $B_i$. The Hammersley–Clifford theorem [98] gives the general form for a distribution with these Markov properties:

$$P(x) = \prod_{c \in C} F_c(x)$$  \hspace{1cm} (1.38)

where $C$ is the set of maximal cliques of $G$—defined to be the set of maximal subgraphs of $G$ that are fully connected in $E$, and maximal in the sense that no further node can be added to a clique without removing the full connectedness property. Note that for a tree the cliques are simply the edges of the graph, and so the decomposition (1.38) simplifies to the decomposition (1.22) for trees, above. Usually the decomposition is expressed in terms of an energy or cost function $E(x)$:

$$P(x) \propto \exp\left(-E(x)\right) \text{ where } E(x) = \sum_{c \in C} \Psi_c(x).$$  \hspace{1cm} (1.39)

More generally, there may be a dependence on parameters, so that

$$P(x) = \frac{1}{Z(\omega)} \exp\left(-E(x, \omega)\right) \text{ where } E(x, \omega) = \sum_{c \in C} \Psi_c(x, \omega),$$  \hspace{1cm} (1.40)

and now the partition function

$$Z(\omega) = \sum_x \exp\left(-E(x, \omega)\right)$$  \hspace{1cm} (1.41)

is included to maintain the normalization condition for the distribution, $\sum_x P(x) = 1$.

An alternative representation of the undirected graph in figure 1.1 is the factor graph [276]. The undirected graph (or Markov network) makes conditional dependencies explicit, but factorization properties are somewhat implicit—they need to be computed in terms of maximal cliques. Factor graphs are a little more complex in that they introduce a second type of node, the function node, in addition to the nodes for variables but have the advantage of making factorization explicit. In many of the cases used in this book—for example, figure 1.1a, with 4-way connectivity between pixels—the factorization structure is straightforward. Each edge in that example is a maximal clique, and therefore the factors are functions of the two variables on the nodes belonging to each edge. On the other hand,
1 Introduction to Markov Random Fields

Figure 1.1b, with its 8-way connectivity, has a more complex set of statistical dependencies, and the factors may not simply correspond to edges. The most general factor structure for the Markov model with the statistical dependencies denoted by that graph is a function of the four variables in each of the $2 \times 2$ squares of variables, which are the maximal cliques of the graph. In computer vision, it is usual to define models directly in terms of their factors, in contrast to normal practice in statistical modeling, where models are defined first in terms of their Markov properties, with factors specified subsequently over maximal cliques, as in (1.38). In the more complex cases of factors involving more than two variables at a time, factor graphs are useful, and are mentioned in chapters 21 and 24. For the most part, though, pairwise factors and simple undirected graphs suffice.

1.4.1 Example: Pseudo-Boolean Energy on a 4-Connected Graph of Pixels

A standard example of an MRF, just about the simplest one that is interesting, is the Ising model with the single parameter $\omega = \{\gamma\}$, whose origins are in statistical physics [523]. The state-space consists of Boolean variables $x_i \in \mathcal{L} = \{0, 1\}$, and the energy function $E(x)$ is termed a Pseudo-Boolean Function (PBF)—because its input is Boolean but its output is not (the energy is real valued). As for the graph, the maximal cliques are the horizontal and vertical edges of the rectangular graph of pixels shown in figure 1.5a. All cliques in this

\[\begin{array}{c}
\text{(a)}
\end{array}\]

\[\begin{array}{c}
\text{(b)}
\gamma=0.5 \\
\gamma=0.7 \\
\gamma=0.8
\end{array}\]

Figure 1.5
Simulating a simple model: The Ising model. (a) The horizontal and vertical edges of a rectangular grid form the cliques of the Ising model. (b) Simulations of typical probable states of the Ising model for various values of the coherence parameter $\gamma$. 
case are of size 2, containing two nodes (pixels). The clique potentials, referred to in this kind of model as \textit{pairwise} potentials because the cliques are pairs of pixels, are

$$
\Psi_{ij}(x_i, x_j) = \gamma |x_i - x_j|.
$$

(1.42)

This represents a penalty \( \gamma \) that increases the energy \( E \) wherever adjacent \( x_i \) and \( x_j \) have different values, and so reduces the joint probability \( P(\mathbf{x}) \) by a factor \( e^{\gamma'} \). This enhances the probability of configurations \( \mathbf{x} \) in which there is large-scale agreement between the values of adjacent pixels. In fact, a moment’s thought will make it clear that the \textit{total} penalty \( \sum_{(i, j) \in \mathcal{C}} \Psi_{ij} \) is simply \( \gamma \) times the total perimeter (in the Manhattan metric) of boundaries separating regions of value 1 from regions of value 0. Thus the distribution \( P(\mathbf{x}) \) favors configurations \( \mathbf{x} \) in which that total perimeter is relatively small. The simulations in figure 1.5b show how higher values of \( \gamma \) indeed tend to favor larger regions of 1s and 0s.

It is worth saying at this stage, and this will come out repeatedly later, that algorithms for inference with general graphs that contain loops—many loops in the case of the example above—are hard. Even simulation from the Ising model is hard, compared with the equivalent simulation for a chain or tree, which is straightforward and can be done exactly. The simulations of the Ising model above were done using a form of \textit{Markov chain Monte Carlo}, adapted specifically for the Ising model [165], the Swendsen–Wang iterative sampler [25].

1.5 Hidden MRF Models

A Markov random field \( P(\mathbf{X} = \mathbf{x}) \), as in the previous section, can act as a prior model for a set of hidden random variables \( \mathbf{X} \), under a set of observations \( \mathbf{z} \), in direct analogy to the HMM model of section 1.2. As with HMMs, the observations are most simply modeled as random variables that are independent when conditioned on the hidden variables \( \mathbf{X} \). This is illustrated in figure 1.6, in which the simple 4-connected graph of figure 1.5a appears as a layer of hidden variables \( \mathbf{x} \) with observations \( \mathbf{z} \) distributed across sites. Each individual observation \( z_i \) depends statistically just on the state \( x_i \) of the corresponding pixel. Now the posterior for the state \( \mathbf{x} \) of the pixels is obtained from Bayes’s formula, just as it was for HMMs (1.6), as

$$
P(\mathbf{x} | \mathbf{z}, \omega) \propto P(\mathbf{z} | \mathbf{x}, \omega) P(\mathbf{x} | \omega),
$$

(1.43)

with the observation likelihood \( P(\mathbf{z} | \mathbf{x}) \) factorized across sites/pixels as it was earlier (1.9), and including the possibility of multiple models as before. It is common also to express this \textit{posterior MRF} in terms of a sum of energies, generalizing the prior MRF (1.40) to include terms for the observation likelihood:

$$
P(\mathbf{x} | \mathbf{z}, \omega) = \frac{1}{Z(\mathbf{z}, \omega)} \exp - E(\mathbf{x}, \mathbf{z}, \omega),
$$

(1.44)

where \( E(\mathbf{x}, \mathbf{z}, \omega) = \sum_{c \in \mathcal{C}} \Psi_c(\mathbf{x}, \omega) + \sum_i \Phi_i(x_i, z_i) \).

(1.45)
Here $Z(z, \omega)$ is the partition function for the posterior MRF. Unlike the HMM (1.8), for which $Z$ can in fact be computed quite easily, computing the partition function $Z(z, \omega)$ for the posterior MRF is intractable.

The most common form of inference over the posterior MRF in vision and image-processing problems, is Maximum A Posteriori (MAP) estimation. MAP inference of $x$ is done in principle by computing $\hat{x} = \arg \max P(x | z)$, or equivalently by minimizing energy:

$$\hat{x} = \arg \min E(x, z, \omega).$$

(1.46)

Note that this does not require knowledge of the partition function $Z(z, \omega)$, which is just as well, given its intractability. The energy functions for many commonly used Markov models (see examples below) can be written as a sum of unary and pairwise terms:

$$E(x, z, \omega) = \sum_{i \in V} \Phi_i(x_i, z_i, \omega) + \sum_{(i,j) \in E} \Psi_{ij}(x_i, x_j, \omega).$$

(1.47)

Algorithms for computing the MAP are discussed in the following sections. Suffice it to say that the MAP can in fact be computed exactly in a time that is polynomial with respect to the size $N$ of the image array, using graph cut, at least when the prior $P(x)$ is an Ising distribution.

### 1.5.1 Example: Segmentation on a 4-Connected Graph of Pixels

Here we give the simplest useful example of a hidden MRF model, for segmenting an image into foreground and background components. The state-space is Boolean, so $x_i \in \{0, 1\}$ denotes background/foreground labels, respectively. The model, originated by Boykov and
Jolly [66], has a number of variants. For tutorial purposes the simplest of them is illustrated here. It uses the Ising model as a prior to encourage the foreground and background components to be as coherent as possible. Thus the $\Psi$ terms in the hidden MRF model (1.45) are the ones from the Ising prior (1.42). The likelihood terms (for details, see chapter 7 on MRF models for segmentation) can be specified by constructing histograms $h_F(z)$ and $h_B(z)$ in color space for foreground and background, respectively (taking care to avoid zeros), and setting

$$\Phi_i(z_i) = \log h_F(z_i) - \log h_B(z_i).$$  

(1.48)

The resulting model specifies a posterior, which is maximized to obtain the estimated segmentation $\hat{x}$, and the resulting method is demonstrably effective, as figure 1.7 shows.

---

**Figure 1.7**

MRF model for bilevel segmentation. (a) An image to be segmented. (b) Foreground and background regions of the image are marked so $x_i$ in those regions is no longer hidden but observed. The problem is to infer foreground/background labels in the remaining unlabeled region of the trimap. (c) Using simply a color likelihood model learned from the labeled regions, without the Ising prior, the inferred labeling is noisy. (d) Also introducing a pairwise Ising term, and calculating the MAP estimate for the inferred labels, deals substantially with the noise and missing data. (Results of the CRF variant of the Ising term, described below, are illustrated here.)
Figure 1.8
MRF model for image reconstruction. (a) An image with added noise, and a portion masked out. (b) Introducing a truncated quadratic prior and calculating (approximately) the MAP estimate of the hidden variables deals much more effectively with the noise.

1.5.2 Example: Image Reconstruction
A classic example of a multilabel MRF problem is image reconstruction, in which a noisy and/or damaged image requires repair. An example is shown in figure 1.8. In this case the state-space may have $x_i \in \{0, 255\}$, corresponding to the possible gray values of pixels in the reconstructed image. A suitable model for reconstruction is to choose the unary as follows:

$$\Phi_i(x_i) = \begin{cases} (x_i - z_i)^2 & \text{if } z_i \text{ observed} \\ 0 & \text{otherwise.} \end{cases}$$ (1.49)

The pairwise term is chosen to encourage smoothness in the reconstructed image, but not so strongly as to blur across genuine edges. A suitable choice is a truncated quadratic prior of the form

$$\Psi(x_i, x_j) = \lambda \min((x_i - x_j)^2, \Psi_{\max}).$$ (1.50)

(More detail is given in later chapters, particularly chapter 11.) The special case that $\Psi_{\max} = 1$ gives the classic Potts model, which penalizes any nonzero difference between $x_i$ and $x_j$ equally, regardless of magnitude, with $\lambda$.

1.5.3 Continuous Valued MRFs
So far we have seen examples of MRFs with discrete label values—either Boolean or multivalued: for example, integer. Of course it is natural in many cases to regard hidden variables as continuous. For example, the underlying image in an image reconstruction problem is a physical property of the world, and its values are most naturally regarded as continuous. Visual reconstruction is therefore often cast in terms of continuous variables [172, 54],...
and in this book, chapters 8, 12, and 13 deal with MRFs over continuous variables. One direct approach is to define Gaussian distributions over the variables, so-called GMRFs, as in chapter 13. Nonlinear variations on that basic theme also have interesting and useful properties.

1.5.4 Conditional Random Field

A Conditional Random Field (CRF) is a form of MRF that defines a posterior for variables \( x \) given data \( z \), as with the hidden MRF above. Unlike the hidden MRF, however, the factorization into the data distribution \( P(x|z) \) and the prior \( P(x) \) is not made explicit [288]. This allows complex dependencies of \( x \) on \( z \) to be written directly in the posterior distribution, without the factorization being made explicit. (Given \( P(x|z) \), such factorizations always exist, however—infinitely many of them, in fact—so there is no suggestion that the CRF is more general than the hidden MRF, only that it may be more convenient to deal with.) One common application of the CRF formalism in vision is in Boolean (foreground/background) segmentation, where it is natural to think of a modification of the Ising prior (1.42) to incorporate some data dependency [65]:

\[
\Psi_{ij}(x_i, x_j, z) = f(z)\gamma|x_i - x_j|, \tag{1.51}
\]

in which the additional term \( f(z) \leq 1 \) weakens the penalty \( \gamma \) wherever the image data suggest the presence of a segmentation boundary—for instance, where image contrast is high. This is described in detail in chapter 7. CRFs are also used in scene labeling [284], for example, to label areas of a scene as natural or as man-made. Other applications of CRFs appear in chapters 9, 11, 24, 25, and 27.

1.6 Inference: MAP/Marginals

Given a hidden MRF for some posterior distribution \( P(x | z) \), as in (1.44) but omitting the parameters \( \omega \), the common inference problems are to estimate either the most probable state

\[
\arg \max_x P(x | z) \tag{1.52}
\]

or the marginal distributions at each pixel, \( P(x_i | z), \ i = 1, \ldots, N \). This is in direct analogy to MAP inference and inference of marginals for HMMs, as discussed in section 1.2, but closed form algorithms are no longer available for two-dimensional problems. Dropping the explicit mention of the data \( z \), the inference problems are equivalent to estimating the mode and marginals of an MRF \( P(x) \), as in (1.39). The remainder of this section outlines various approaches to estimation using this posterior distribution.

1.6.1 Gibbs Sampling

Gibbs sampling is a procedure introduced by Geman and Geman [161] for sampling fairly from an MRF. At successive visits to each site \( i \), the variable \( x_i \) is sampled from the local,
conditional distribution \( P(x_i \mid \{x_j, (i, j) \in E, j \neq i\}) \), and all sites are visited, arbitrarily often, in some random order. Asymptotically, after many visits to each site, the set of samples \((x_1, \ldots, x_N)\) settles down to be a fair sample from the MRF. However, this burn-in process may happen very slowly, and this is a problem for practical application. Further details of Gibbs sampling are given in chapter 5.

### 1.6.2 Mean Field Approximation
Mean field approximation is a form of variational approximation in which the distribution \( P(x) \) is approximated as a product of factors,

\[
P(x) \approx \prod_i b_i(x_i),
\]

(1.53)

and the factors become the approximations to the marginals \( P(x_i) \) of the distribution. Of course this factorized form cannot represent the posterior exactly, but mean field algorithms choose the \( b_i \) in such a way as to approximate the posterior as closely as possible. This is done by minimizing KL divergence, a single numerical measure of difference between the true and approximate distributions. Full details of this important approach are given in chapter 5.

**Classical Algorithms for MAP Inference** The MAP solution of a hidden Markov model can be found by minimizing an energy function (1.45). A number of algorithms exist for solving these minimization problems defined over discrete or continuous random variables. Some of the best-known of these algorithms will be reviewed in this section. Comparisons of their performance on various test problems will be given in chapter 11.

### 1.6.3 Iterated Conditional Modes (ICM)
Iterated Conditional Modes (ICM) are one of the oldest and simplest MAP inference algorithms [38, 235]. They belong to the family of local (also called move-making) algorithms that start with an initial solution. At each step these algorithms explore a space of possible changes (also called a move space) that can be made to the current solution \( x^c \), and choose a change (move) that leads to a new solution \( x^n \) having the lowest energy. This move is referred to as the optimal move. The algorithm is said to converge when no solution with a lower energy can be found.

ICM works in a coordinate descent fashion. At each step it chooses a variable \( X_i, i \in V \). Keeping the values of all other variables fixed, it finds the label assignment for \( X_i \) that leads to the lowest energy:

\[
x^n_i = \arg \min_{x_i \in L} E(\{x^c_j : j \neq i\}, x_i).
\]

(1.54)

This process is repeated for other variables until the energy cannot be decreased further (i.e., all variables are assigned labels that are locally optimal.) Note that the descent step is efficient in that it need involve only variables in the Markov blanket of \( i \).
1.6.4 Simulated Annealing

The ICM algorithm makes changes to the solution if they lead to solutions having lower energy. This greedy characteristic makes it prone to getting stuck in local minima. Intuitively, an energy minimization algorithm that can jump out of local minima will do well in problems with a large number of local optima. Simulated annealing (SA) is one such class of algorithms. Developed as a general optimization strategy by Kirkpatrick et al. [234], simulated annealing allows changes to the solution that lead to a higher energy with a certain probability. This probability is controlled by a parameter $T$ that is called the temperature. A higher value of $T$ implies a high probability of accepting changes to the solution that lead to a higher energy. The algorithm starts with a high value of $T$ and reduces it to 0 as it proceeds. When $T = 0$ the algorithm accepts only changes that lead to a decrease in energy, as, for example, ICM does. Details are given in chapter 5.

1.6.5 Loopy Belief Propagation

We have seen how the sum-product and max-product message-passing algorithms can be used to perform inference in tree structured graphs. Although these algorithms are guaranteed only to find the optimal solutions in tree structured graphs, they have been shown to be effective heuristics for MAP/marginal estimation even in graphs with loops, such as the grid graphs found in computer vision.

The max-product algorithm can be used to minimize general energy functions approximately. Since an energy function is the negative log of posterior probability, it is necessary only to replace the product operation with a sum operation, and the max operation with a min. For the pairwise energy function (4.1) the message $m_{j \rightarrow i}$ from a variable $X_j$ to any neighboring variable $X_i$ is computed as

$$m_{j \rightarrow i}(x_i) = \min_{x_j} \left( \Phi_j(x_j) + \Psi_{ij}(x_i, x_j) + \sum_{k \in \mathcal{N}(j) - \{i\}} m_{k \rightarrow j}(x_j) \right). \quad (1.55)$$

These messages can also be used to compute a (min-sum) analogue of the belief, called the min-marginal:

$$M(x_i) = \min_{x - \{x_i\}} E(x). \quad (1.56)$$

The min-marginal can be computed from the messages as

$$M^*(x_i) = \Phi_i(x_i) + \sum_{k \in \mathcal{N}(j) - \{i\}} m_{k \rightarrow j}(x_j). \quad (1.57)$$

The min-marginal can be used to find an alternative estimate to the MAP assignment of a variable, as follows:

$$x_i^* = \arg \min_{x_i} M^*(x_i). \quad (1.58)$$
Chapter 10 will describe an application of Bayesian belief propagation to quickly find approximate solutions to two-dimensional vision problems, such as image superresolution using patch priors, and for the problem of transferring the style of a picture (e.g., a painting) to another picture (e.g., a photograph). Those results have inspired a lot of work on the theoretical properties of the way in which BP operates in a general probabilistic model [517]. For instance, it was shown that when sum-product BP converges, the solution is a local minimum of the Bethe approximation [539] of the free energy defined by the Markov model. More recently, a number of researchers have shown the relationship between message-passing algorithms and linear programming relaxation-based methods for MAP inference [504, 536]. This relationship is the subject of chapter 6.

1.7 MAP Inference in Discrete Models

Many problems in vision and machine learning give rise to energies defined over discrete random variables. The problem of minimizing a function of discrete variables is in general NP-hard, and has been well studied in the discrete optimization and operations research communities. A number of “greedy” and “approximate” techniques have been proposed for solving these discrete optimization problems. An introduction to these methods is given in this section.

Although minimizing a function of discrete variables is NP-hard in general, there are families of energy functions for which it can be done in polynomial time. Submodular functions are one such well-known family [155, 324]. In some respects they are analogous to the convex functions encountered in continuous optimization, and have played a big part in the development of efficient algorithms for estimating the MAP solution of many image labeling problems. For instance, the Ising energy described earlier (1.42) is an important example in vision and image processing of a submodular function.

It is worth reflecting for a moment that this approach gives an exact solution to inference problems at image scale. This is quite remarkable, given how rare it is that realistic scale information problems in the general area of machine intelligence admit exact solutions. Graph cut was first used to give exact solutions for Boolean MRF problems over images by Greig et al. [176], at a time when the method was so slow that it could be used only to benchmark faster, approximate algorithms. Since then, the progress in understanding and developing graph cut algorithms in general, and for images specifically, allows these algorithms to be regarded as highly practical. Even real-time operation is possible, in which several million pixels in an image or video are processed each second.

1.7.1 Submodular Pseudo-Boolean Functions

The minimization of a Pseudo-Boolean Function (PBF) \( E : \{0, 1\}^n \rightarrow \mathbb{R} \) is a well-studied problem in combinatorial optimization [215] and operations research [61]. A PBF \( f : \{0, 1\}^n \rightarrow \mathbb{R} \) is submodular if and only if, for all label assignments \( \mathbf{x}_A, \mathbf{x}_B \in \{0, 1\}^n \), the function satisfies the condition
\[ f(x_A) + f(x_B) \geq f(x_A \lor x_B) + f(x_A \land x_B), \]  
\[ \text{where } \lor \text{ and } \land \text{ are componentwise OR and AND, respectively. From the above definition it can easily be seen that all PBFs of arity 1 are submodular. Similarly, any PBF of arity 2 is submodular if and only if} \]
\[ f(1, 0) + f(0, 1) \geq f(1, 1) + f(0, 0). \]  

Another interesting property of submodular functions is that the set of submodular functions is closed under addition (i.e., the sum of two or more submodular functions is another submodular function). This condition implies that the energy of a pairwise MRF (1.47) is submodular if all the pairwise potentials \( \Psi_{ij} \) are submodular. For example, the Ising model pairwise potential consists of terms \( \Psi_{ij}(x_i, x_j) = \gamma |x_i - x_j| \) (1.42), which are submodular, as is apparent from substituting \( f(x, x') = \gamma |x - x'| \) into (1.60). Hence the entire Ising potential function is submodular.

### 1.7.2 Minimizing Submodular Pseudo-Boolean Functions Using Graph Cut

The first polynomial time algorithm for minimizing submodular functions [215, 427] had high practical runtime complexity. Although recent work has been partly successful in reducing the complexity of algorithms for general submodular function minimization, they are still quite expensive computationally and cannot practically be used for large problems. For instance, one of the best algorithms for general submodular function minimization has a worst case complexity \( O(n^5 Q + n^6) \), where \( Q \) is the time taken to evaluate the function [358]. This is certainly too expensive for use in computer vision, where \( n \) is frequently of the order of millions (of pixels).

However, there is one important subclass of submodular functions that can be optimized much more efficiently: the submodular functions of arity at most 2, corresponding to MRFs with factors that are functions of at most two variables. This is the common case that was illustrated in the graphs in figure 1.1. Optimization over this class is known as Quadratic Pseudo-Boolean Optimization or (QPBO). It turns out to be equivalent to finding the minimum cost cut in a certain graph [61, 68, 211, 421] — the so-called s-t min-cut problem. This is described in detail in chapter 2. It leads to efficient algorithms for finding the MAP solution of many important pairwise MRF models for vision problems, for example, foreground/background image segmentation problems like the one illustrated in figure 1.7.

A general QPB function can be expressed as:

\[
E(x) = \theta_{\text{const}} + \sum_{i \in V} (\theta_{i:1} x_i + \theta_{i:0} x_i) \\
+ \sum_{(i, j) \in E} (\theta_{ij:11} x_i x_j + \theta_{ij:01} x_i x_j + \theta_{ij:10} x_i x_j + \theta_{ij:00} x_i x_j),
\]

where \( \bar{x}_i \) defines the complementary variable \( \bar{x}_i = 1 - x_i \). Parameter \( \theta_{i:a} \) is the penalty for assigning label \( a \) to latent variable \( x_v \), and \( \theta_{ij:ab} \) is the penalty for assigning labels \( a \) and \( b \)
1 Introduction to Markov Random Fields

Figure 1.9
Energy minimization using graph cut. The figure shows how individual unary and pairwise terms of an energy function taking two binary variables are represented and combined in the graph. Multiple edges between the same nodes are merged into a single edge by adding their weights. For instance, the cost $w_1$ of the edge $(s, x_a)$ in the final graph is equal to $w_1 = \theta_{a;0} + \theta_{ab;00}$. The cost of an $s$-$t$ cut in the final graph is equal to the energy $E(x)$ of the configuration $x$ the cut induces. The minimum cost $s$-$t$ cut induces the least-energy configuration $x$ for the energy function.

to the latent variables $x_i$ and $x_j$, respectively. To minimize this energy with $s$-$t$ min-cut, the individual unary and pairwise terms of the energy function are represented by weighted edges in the graph. Multiple edges between the same nodes are merged into a single edge by adding their weights. The graph construction in the simple case of a two-variable energy function is shown in figure 1.9. In this graph an $s$-$t$ cut is defined to be a curve that separates the source node $s$ from the terminal node $t$. The cost of such a cut is defined to be the sum of the weights of edges traversed by the cut. The $s$-$t$ cut with the minimum cost provides the minimum solution $x^*$ by disconnecting a node $x_i$ either from $s$, representing the assignment $x_i = 0$, or from $t$, representing $x_i = 1$. The cost of that cut corresponds to the energy of the solution $E(x^*)$ modulo the constant term $\theta_{\text{const}}$. Algorithms for finding the $s$-$t$ min-cut require that all edges in the graph have nonnegative weights. This condition results in a restriction that the energy function $E$ be submodular. This is all explained in detail in chapter 2.

1.8 Solving Multilabel Problems Using Graph Cut

Many computer vision problems involve latent variables that can take values in integer spaces. The energy functions corresponding to these problems are defined over multistate variables. Graph cut-based algorithms for minimizing such functions can be divided into two broad categories: transformation methods and move-making algorithms. Transformations methods transform the energy minimization problem defined over multistate variables $X_i \in \mathcal{L} = \{l_1, \ldots, l_k\}, k \geq 2$ to one that is defined over $k$ binary variables per multistate variable. This is done by encoding different states of the multistate variables by the assignments of a
set of Boolean variables—a full discussion is given in chapter 4. Move-making algorithms, on the other hand, work by decomposing the problem into a set of problems defined over Boolean random variables, which, as we have seen, can be solved efficiently using $s$-$t$ min-cut algorithms. In practice, move making has been the dominant approach for multilabel problems.

### 1.8.1 Graph Cut-Based Move-Making Algorithms

The key characteristic of any move-making algorithm is the number of possible changes it can make in any step (also called the size of the move space). A large move space means that extensive changes can be made to the current solution. This makes the algorithm less prone to getting stuck in local minima and also results in a faster rate of convergence. Boykov et al. [72] proposed two move-making algorithms: $\alpha$-expansion and $\alpha\beta$-swap, whose move, space size increases exponentially with the number of variables involved in the energy function. The moves of the expansion and swap algorithms can be encoded as a vector of binary variables $t = \{t_i, \forall i \in V\}$. The transformation function $T(x^c, t)$ of a move algorithm takes the current labeling $x^c$ and a move $t$, and returns the new labeling $x^n$ that has been induced by the move.

The expansion algorithm has one possible move per label $\alpha \in L$. An $\alpha$-expansion move allows any random variable either to retain its current label or to take the label $\alpha$. The transformation function $T_\alpha()$ for an $\alpha$-expansion move transforms the current label $x^c_i$ of any random variable $X_i$ as

$$
x^n_i = T_\alpha(x^c_i, t_i) = \begin{cases} 
\alpha & \text{if } t_i = 0 \\
x^c_i & \text{if } t_i = 1.
\end{cases}
$$  

(1.62)

One iteration of the algorithm involves making moves for all $\alpha$ in $L$ successively in some order.

The swap algorithm has one possible move for every pair of labels $\alpha, \beta \in L$. An $\alpha\beta$-swap move allows a random variable whose current label is $\alpha$ or $\beta$ to take either label $\alpha$ or label $\beta$. The transformation function $T_{\alpha\beta}()$ for an $\alpha\beta$-swap transforms the current label $x^c_i$ of a random variable $X_i$ as

$$
x^n_i = T_{\alpha\beta}(x^c_i, t_i) = \begin{cases} 
\alpha & \text{if } x^c_i = \alpha \text{ or } \beta \text{ and } t_i = 0 \\
\beta & \text{if } x^c_i = \alpha \text{ or } \beta \text{ and } t_i = 1.
\end{cases}
$$  

(1.63)

One iteration of the algorithm involves performing swap moves for all $\alpha$ and $\beta$ in $L$ successively in some order.

The energy of a move $t$ is the energy of the labeling $x^n$ that the move $t$ induces, that is, $E_m(t) = E(T(x^c, t))$. The move energy is a pseudo-Boolean function ($E_m: \{0, 1\}^n \rightarrow \mathbb{R}$) and will be denoted by $E_m(t)$. At each step of the expansion and swap-move algorithms, the optimal move $t^*$ (i.e., the move decreasing the energy of the labeling by the greatest amount) is computed. This is done by minimizing the move energy, that is, $t^* = \arg \min_t E(T(x^c, t))$. Boykov et al. [72] showed that for certain families of pairwise energy functions, the move
energy is submodular, and hence the optimal move $t^*$ can be computed in polynomial time using $s$-$t$ min-cut algorithms. More details on the expansion and swap algorithms can be found in chapter 3.

### 1.9 MAP Inference Using Linear Programming

Linear programming (LP) is a popular method for solving discrete optimization problems. It has also been extensively used for MAP inference in discrete Markov models. To convert the energy minimization problem into an LP, we will first need to formulate it as an integer program (IP). This is illustrated for the pairwise energy function (4.1). The energy function is first linearized using binary indicator variables $y_{i;a}$ and $y_{ij;ab}$ for all $i, j \in \mathcal{V}$ and $l_a, l_b \in \mathcal{L}$. The indicator variable $y_{i;a} = 1$ if $x_i = l_a$, and $y_{i;a} = 0$ otherwise. Similarly, the variables $y_{ij;ab}$ indicate the label assignment $x_i = l_a, x_j = l_b$. The resulting IP can be written as

\[
\text{Minimize} \quad \sum_{i \in \mathcal{V}, l_a \in \mathcal{L}} \Phi_i(l_a) y_{i;a} + \sum_{(i,j) \in \mathcal{E},\ l_a, l_b \in \mathcal{L}} \Psi_{ij}(l_a, l_b) y_{ij;ab}, \tag{1.64}
\]

subject to

\[
\sum_{l_a \in \mathcal{L}} y_{ij;ab} = y_{j;b}, \quad \forall (i, j) \in \mathcal{E}, l_b \in \mathcal{L}, \tag{1.65}
\]

\[
\sum_{l_b \in \mathcal{L}} y_{ij;ab} = y_{i;a}, \quad \forall (i, j) \in \mathcal{E}, l_a \in \mathcal{L}, \tag{1.66}
\]

\[
\sum_{l_a \in \mathcal{L}} y_{i;a} = 1, \quad \forall i \in \mathcal{V}, \tag{1.67}
\]

\[
y_{i;a}, y_{ij;ab} \in \{0, 1\} \quad \forall i \in \mathcal{V}, \forall (i, j) \in \mathcal{E}, \forall l_a, l_b \in \mathcal{L}. \tag{1.68}
\]

The constraint (1.65) enforces consistency over marginalization, and constraint (1.67) makes sure that each variable is assigned only one label.

Relaxing the integrality constraints (1.68) of the IP leads to an LP problem that can be solved in polynomial time using general-purpose LP solvers. These solvers are relatively computationally expensive, and make this approach inappropriate for vision problems that involve a large number of variables. A number of message-passing and maximum flow-based methods have recently been developed to efficiently solve the LP defined above [270, 277, 504, 520]. See chapters 6 and 17 for more details, including discussion of when the solution to the LP is also a solution to the IP.

#### 1.9.1 Nonsubmodular Problems in Vision

We have seen how submodular pseudo-Boolean functions can be minimized using algorithms for computing the $s$-$t$ min-cut in weighted graphs. Although this is an extremely
useful method, energy functions corresponding to many computer vision problems are not submodular. We give two examples here by way of motivation. The first is the fusion move described in chapter 18, in which two trial solutions to a problem such as optimal image denoising are available, and a final solution is to be constructed by combining the two, pixel by pixel. A binary array is constructed that switches each pixel between its values in the first and second solutions. The original denoising functional now induces a functional over the Boolean array that will not, in general, be submodular (see chapter 18 for details). A second example of an important nonsubmodular problem arises in the general case of multilabel optimization, as discussed in the previous section. (Details are given in chapter 3.)

Minimizing a general nonsubmodular energy function is an NP-hard problem. We also saw earlier how general energy minimization problems can be formulated in terms of an IP. Relaxing the integrality constraints of the IP leads to an LP problem, and this is attractive because there is no requirement for the IP energy to be submodular. Chapter 2 will explain a particular relaxation of QPBO called the roof dual [61, 261], which leads to an LP that can be solved efficiently using s-t min-cut algorithms.

1.10 Parameter Learning for MRFs

Learning the parameters $\omega$ of an MRF (or CRF) $P(x|z, \omega)$ from labeled training data is an important problem. The alternative is to attempt to select the parameters $\omega$ by hand or by experiment; this is known to be difficult, and quite impractical if the dimensionality of $\omega$ is at all large. It also proves to be a challenging problem. Note that here the learning problem is rather different from the parameter learning problem for HMMs discussed earlier. It is a little easier in that the data are labeled—that is, the $x$-values are known for each $z$ in the training set—those previously hidden variables $x$ in the model may become observed variables in the context of learning. In contrast, in HMM parameter learning, the state values $x$ were unknown. But it is a harder problem in that the estimation is done over a graph that is not merely a tree, so the underlying inference problem is not exactly tractable.

1.10.1 Maximum Likelihood

A standard approach to parameter learning is via Maximum Likelihood Estimation (MLE), that is, by solving

$$\max_\omega L(\omega) \text{ where } L(\omega) = P(x|z, \omega).$$

(1.69)

Now, from (1.40),

$$\log L(\omega) = -\log Z(\omega) - \sum_{c \in C} \Psi_c(x_c, \omega),$$

(1.70)

and differentiating this to maximize the likelihood $L$ w.r.t. $\omega$ is entirely feasible for the second term, as it is decomposed into a sum of local terms, but generally intractable for the
first term, the log partition function that was defined earlier (1.41). This is well known, and is fully described in standard texts on Markov random fields [523, 309].

As a result, it is necessary to approximate the likelihood function in order to maximize it. Probably the best-known classical approximation is pseudo likelihood in which \( L \) is replaced by

\[
L^* = \prod_i P(x_i \mid \{x_j, (i, j) \in E\}),
\]

(1.71)
a product of local, conditional densities. Its log is a sum of local functions, which can tractably be differentiated in order to maximize with respect to parameters. This pseudo likelihood function does not itself approximate the likelihood \( L \), but its maximum is known to approximate the maximum of \( L \) under certain conditions [523].

Alternative approximate learning schemes have been proposed by others. For example, Kumar et al. [283] propose three different schemes to approximate the gradients of the log likelihood function in the case of pairwise energy functions, such as the one used for image segmentation (1.47). The derivatives of the log likelihood are easily shown to be a function of the expectations \( \langle x_i \rangle_{\omega, z} \). If we were given the true marginal distributions \( P(x_i \mid z, \omega) \), we could compute the exact expectations:

\[
\langle x_i \rangle_{\omega, z} = \sum_{x_i} x_i P_i(x_i \mid z, \omega).
\]

(1.72)

This is generally infeasible, so the three proposed approximations are as follows.

1. **Pseudo Marginal Approximation (PMA)** Pseudo marginals obtained from loopy belief propagation are used instead of the true marginals for computing the expectations.

2. **Saddle Point Approximation (SPA)** The label of the random variable in the MAP solution is taken as the expected value. This is equivalent to assuming that all the mass of the distribution \( P_i(x_i \mid z, \theta) \) is on the MAP label.

3. **Maximum Marginal Approximation (MMA)** In this approximation the label having the maximum value under the pseudo marginal distribution obtained from BP is assumed to be the expected value.

### 1.10.2 Max-Margin Learning

An alternative approach to parameter learning that has become very popular in recent years is max-margin learning. Similar to ML estimation, max-margin learning also uses inference in order to compute a gradient with respect to the parameters; however, only the MAP labeling need be inferred, rather than the full marginals.

Most margin-based learning algorithms proposed and used in computer vision are inspired from the structured support vector machine (SVMstruct) framework of Tsochantaridis et al. [484] and the maximum margin network learning of Taskar et al. [14, 473]. The goal
of these methods is to learn the parameters so that the ground truth has the lowest energy
by the largest possible margin \( \Theta \) or, if that is not possible, that the energy of the ground
truth is as close as possible to that of the minimum energy solution. More formally,

\[
\max_{\omega:|\omega| = 1} \Theta \quad \text{such that} \quad E(x, z; \omega) - E(\hat{x}, z; \omega) \geq \Theta \quad \forall x \neq \hat{x},
\]

where \( \hat{x} \) is the MAP estimate for \( x \). A detailed discussion of max-margin learning for MRFs
is given in chapter 15.

This concludes the introduction to Markov random fields. We have tried to set out the
main ideas as a preparation for the more detailed treatment that follows. In part I of the book,
some of the main algorithms for performing inference with MRFs are reviewed. Then in part
II, to reward the reader for hard work on the algorithms, there is a collection of some of the
most successful applications of MRFs, including segmentation, superresolution, and image
restoration, together with an experimental comparison of various optimization methods on
several test problems in vision. Part III discusses some more advanced algorithmic topics,
including the learning of parameter values to tune MRF models, and some approaches to
learning and inference with continuous-valued MRFs. Part IV addresses some of the limita-
tions of the strong locality assumptions in the small-neighborhood MRFs discussed in this
introduction and in many of the earlier chapters of the book. This includes going beyond
pairwise functions in the MRF factorization, to ternary functions and higher, and to models
that, though sparse, do not restrict neighborhoods to contain pixels that are nearby in the
image array. Finally, part V is a showcase of some applications that use MRFs in more
complex ways, as components in bigger systems or with multiterm energy functions, each
term of which enforces a different regularity of the problem—for instance, spatial and
temporal terms in multiview video, or acting over multiple layers of hidden variables for
simultaneous recognition and segmentation of objects.

1.11 Glossary of Notation

Some of the basic notation for Markov random fields is listed below. It is the notation used
in this introductory chapter, and also is used frequently in later chapters of the book.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{G} = \mathcal{V}, \mathcal{E} )</td>
<td>graph of MRF nodes (sites) ( \mathcal{V} ) and edges ( \mathcal{E} )</td>
</tr>
<tr>
<td>( i \in \mathcal{V} )</td>
<td>index for nodes/sites</td>
</tr>
<tr>
<td>( (i, i') \in \mathcal{E} \text{ or } (i, j) \in \mathcal{E} )</td>
<td>edges linking nodes of ( \mathcal{G} )</td>
</tr>
<tr>
<td>( c \in \mathcal{C} )</td>
<td>cliques of ( \mathcal{G} ), so each ( c \subset \mathcal{V} )</td>
</tr>
<tr>
<td>( z = (z_1, \ldots, z_i, \ldots, z_N) )</td>
<td>image data, pixel value (monochrome or color) at site ( i )</td>
</tr>
<tr>
<td>( \mathbf{X} = (X_1, \ldots, X_i, \ldots, x_N) )</td>
<td>random state variables at site (pixel) ( i )</td>
</tr>
</tbody>
</table>
$x = (x_1, \ldots, x_i, \ldots, x_N)$, values taken by state variables, that is, $X_i = x_i$

$L = \{l_1, \ldots, l_k, \ldots, l_K\}$, label values for discrete MRF, so $x_i \in L$

$P(X | z)$, posterior probability distribution for state given data

$P(x | z)$, shorthand for $P(X = x | z)$

$P(x | z) = \frac{1}{Z(z)} \exp -E(x, z)$, Gibbs energy form of the MRF

$Z(\omega)$ or $Z(\omega, z)$, partition function in Gibbs form of MRF

$E(x) = U(x) + V(x)$, Gibbs energy: unary and neighbor terms

$E(x, z) = U(x, z) + V(x, z)$, Gibbs energy where dependence on data $z$ is explicit

$\omega \in \Omega$, $P(x | z, \omega), E(x, z, \omega)$, Parameters $\omega$ for MRF model

$\Phi_i(x_i)$, unary potential for MRF, so $U(x) = \sum_i \Phi_i(x_i)$

$\Psi_{ij}(x_i, x_j)$, pairwise potentials, so $V(x) = \sum_{ij} \Psi_{ij}(x_{ij})$

$\Psi_c(x_c)$, higher-order potentials,

$V(x) = \sum_{c \in C} \Psi_c(x_c)$, general form of Gibbs term as a sum over cliques

$y_i$, auxiliary Boolean variables, for example, for label expansion schemes
Part I of this book deals with some of the fundamental issues concerning inference in the kinds of MRFs that are used in vision. In machine learning generally, inference is often taken to refer to inference of an entire probability distribution \( P(x \mid z) \) or projections of that distribution as marginal distributions \([46]\) over subsets of variables \( x_i \). This is a general view of inference that allows the output of the inference process to be treated as an intermediate result capable of refinement by fusion with further information as it subsequently may become available. In principle this approach can be taken with images where \( x \) is an array of pixels or some other high-dimensional array of image-related variables such as superpixels.

In practice, for images the dimension of \( x \) is so high that it is quite infeasible to represent the full posterior distribution. One approach is to use Monte Carlo samplers, as described in chapter 1, but it is unlikely that “burn-in” can be achieved on any practical timescale. For low-dimensional information extracted from images, such as curves, it may be practical to represent the full posterior, either exactly or via samplers \([51]\). This book, however, is mostly concerned with inferences over the whole image, and the most that can practically be done is to infer (approximate) marginals over individual pixels or, conceivably, small sets of pixels. This idea is mentioned in chapter 1 and, in more detail, in chapter 5. There are some possible applications for pixelwise marginals in parameter learning (discussed in part III), but on the whole they have not been greatly used in vision. This discussion is all by way of explaining why part I, and indeed much of the book, restricts inference to MAP estimation.

As pointed out in chapter 1, maximizing the posterior \( P(x \mid z) \) is equivalent to minimizing energy \( E(x, z) \), so part I begins with chapter 2 explaining in detail the idea, introduced in chapter 1, of solving pseudo-Boolean optimization exactly, using a graph cut algorithm. The basic graph cut mechanism that gives exact solutions for the minimization of submodular, pseudo-Boolean functions can also give exact solutions when even the Boolean domain is replaced by a multilabel domain such as the integers. The conditions under which this can be done place tight restrictions on the objective function \( E(x) \), and are detailed in chapter 4. Under less strict conditions on \( E(x) \), pseudo-Boolean graph cut can be used as an exact partial optimization step—a move (see chapter 1)—to solve a multilabel optimization
problem. However, the overall solution will be only approximately optimal, as chapter 3 explains.

Graph cut is one of the dominant approaches for inference in MRFs for vision; two others are Loopy Belief Propagation (LBP) and mean field approximation, which were introduced briefly in chapter 1. Chapter 5 describes both methods and explains that they are related to energy minimization, but with a different kind of objective functions known as a free energy. As an alternative to these deterministic methods, stochastic estimation using Markov Chain Monte Carlo (MCMC) is also explained. The final chapter in part I, chapter 6, expands on the idea of Linear Programming (LP) relaxation as a means of embedding an integer-valued optimization problem on a graph in a more tractable, continuous-valued optimization. Interestingly, it turns out that this approach is closely related to BP and that variants of BP can be used as efficient means of solving the LP.
This chapter describes efficient methods for solving an inference problem for pairwise MRF models (equation 1.55) in a simple special case when state variables \( x_i \) are binary. Many problems in computer vision can be represented by such binary models (see part III). One basic example is segmentation of an image into object and background pixels, as in chapter 7. The inference problem can be formulated as finding a binary labeling \( x \) in which the energy \( E \) defined in chapter 1 (equation 1.47) achieves its minima for some given observations \( z_i \) and for some fixed parameter \( \omega \). This section describes efficient optimization techniques for binary pairwise models using standard graph cut algorithms from combinatorial optimization.

This chapter uses a different representation of energy \( E \) that is more convenient for binary variables \( x \). This representation is common in combinatorial optimization literature in which algorithms for binary pairwise energies have been actively studied for forty years; for example, see the survey in [61]. Without loss of generality, assume that binary state variables \( x_i \) take two values, 0 and 1. Then, energy \( E \) can be written as a quadratic pseudo-Boolean function, as in (2.1):

\[
E(x) = \theta_{\text{const}} + \sum_{i \in V} (\theta_{i;1} x_i + \theta_{i;0} \bar{x}_i) \\
+ \sum_{(i,j) \in E} (\theta_{ij;11} x_i x_j + \theta_{ij;01} \bar{x}_i x_j + \theta_{ij;10} x_i \bar{x}_j + \theta_{ij;00} \bar{x}_i \bar{x}_j),
\]

(2.1)

where \( \bar{x}_i = 1 - x_i \) denotes the negation of variable \( x_i \in \{0, 1\} \). “Boolean” refers to the fact that variables \( x_i \) can take only two values, and “quadratic” means that there are only unary (e.g., \( \theta_{i;1} x_i \)) and quadratic (e.g., \( \theta_{ij;11} x_i x_j \)) terms. For binary-valued labelings the \( x \)-function (equation 1.47) is equivalent to (2.1) with constants \( \theta_{i;1} = \Phi_i(1, z_i, \omega), \theta_{i;0} = \Phi_i(0, z_i, \omega), \theta_{ij;11} = \Psi_{ij}(1, 1, z_i, z_j, \omega), \theta_{ij;01} = \Psi_{ij}(0, 1, z_i, z_j, \omega), \theta_{ij;10} = \Psi_{ij}(1, 0, z_i, z_j, \omega), \theta_{ij;00} = \Psi_{ij}(0, 0, z_i, z_j, \omega) \), assuming that observations \( z \) and parameters \( \omega \) are known. Note that the energy (2.1) can also be written as

\[
E(x) = \theta_{\text{const}} + \sum_{i \in V} \theta_{i;x_i} + \sum_{(i,j) \in E} \theta_{ij;x_i x_j}.
\]

(2.1')
In this chapter we will see that for an important class of pseudo-Boolean functions the minimization problem (equation 1.1) can be reduced to a min-cut/max-flow problem on graphs. This is a classical combinatorial optimization problem with many applications even outside of computer vision. It can be solved in polynomial time, and many efficient algorithms have been developed since 1956 [146]. New fast algorithms for the min-cut/max-flow problem are actively being researched in the combinatorial optimization community [11]. Existing algorithms vary in their theoretical complexity and empirical efficiency on different types of graphs. Some efficient algorithms were developed specifically for sparse grids common in computer vision [68, 120].

This chapter describes the general $s$-$t$ min-cut problem and standard algorithms for solving it, and shows how they can be used to minimize pseudo-Boolean functions like (2.1). The structure is as follows:

- Section 2.1 covers terminology and other basic background material on the general $s$-$t$ min-cut problem required to understand this and several later chapters. This section also gives an overview of standard combinatorial algorithms for the $s$-$t$ min-cut problem, focusing on methods known to be practical for regular grid graphs common in vision.

- Section 2.2 describes how graph cut algorithms can be used to globally minimize a certain class of pseudo-Boolean functions (2.1). This class is characterized by a certain submodularity condition for parameters of energy (2.1). Many interesting problems in vision, such as object/background segmentation (as in chapter 7), are based on submodular binary models.

- Section 2.3 concludes this chapter by describing more general graph, cut techniques that can be applied to nonsubmodular binary models. Such methods are not guaranteed to find the solution for all variables, but those variables $x_i$ whose values are determined are guaranteed to be a part of some globally optimal vector $x$. In many practical problems, such partial solutions cover most of the state variables.

2.1 Algorithms for Min-Cut/Max-Flow Problems

2.1.1 Background on Directed Graphs

Chapter 1 described undirected graphs consisting of a set of vertices $\mathcal{V}$, typically corresponding to image pixels, and a set of undirected arcs or edges $\mathcal{E}$, corresponding to a 4, 8, or any other neighborhood system (see figure 1.1). In this chapter we will use the corresponding directed weighted graphs $(\hat{\mathcal{V}}, \hat{\mathcal{E}}, w)$, including two additional terminal vertices, source $s$ and sink $t$,

$$\hat{\mathcal{V}} := \mathcal{V} \cup \{s, t\},$$

(2.2)
and a larger set of directed edges,
\[ \mathcal{E} := \{(s \to i), (i \to t) \mid i \in \mathcal{V}\} \cup \{(i \to j), (j \to i) \mid (i, j) \in \mathcal{E}\} \tag{2.3} \]

whose weights (capacities) are nonnegative: \( w_{ij} \geq 0 \) for \( (i \to j) \in \mathcal{E}. \)

In the context of vision, terminals often correspond to labels that can be assigned to pixels. In figure 2.1a we show a simple example of a two-terminal directed graph that can be used for optimizing the posterior distribution (equation 1.6) on a \( 3 \times 3 \) image in the case of two labels. The structure of graphs representing different problems in computer vision may vary. However, many of them use 2D or 3D grid graphs like the one in figure 2.1a. This is a simple consequence of the fact that graph nodes often represent regular image pixels or voxels.

All directed edges in the graph are assigned some weight or capacity. The cost \( w_{ij} \) of directed edge \( (i \to j) \) may differ from the cost \( w_{ji} \) of the reverse edge \( (j \to i) \). In fact, the ability to assign different edge weights for \( (i \to j) \) and \( (j \to i) \) is important for many applications in vision.

It is common to distinguish two types of edges: n-links and t-links. The n-links connect pairs of neighboring pixels or voxels:
\[ \{(i \to j), (j \to i) \mid (i, j) \in \mathcal{E}\} \quad \text{(n-links)}. \]

---

1. We assume throughout the chapter that the set \( \mathcal{E} \) does not have parallel edges, and that \( (i, j) \in \mathcal{E} \) implies \( (j, i) \notin \mathcal{E} \).
Thus, they represent a neighborhood system in the image. In the context of computer vision, the cost of n-links may correspond to penalties for discontinuities between pixels. t-links connect pixels with terminals (labels):

\[(s \rightarrow i), (i \rightarrow t) \mid i \in \mathcal{V}\] (t-links).

The cost of a t-link connecting a pixel and a terminal may correspond to a penalty for assigning the corresponding label to the pixel.

2.1.2 Min-Cut and Max-Flow Problems

An s-t cut \( C \) on a graph with two terminals \( s, t \) is a partitioning of the nodes in the graph into two disjoint subsets \( S \) and \( T \) such that the source \( s \) is in \( S \) and the sink \( t \) is in \( T \). For simplicity, throughout this chapter we refer to s-t cuts as just cuts. Figure 2.1b shows an example of a cut. In combinatorial optimization the cost of a cut \( C = (S, T) \) is defined as the sum of the costs of “boundary” edges \((i \rightarrow j)\) where \( i \in S \) and \( j \in T \). Note that the cut cost is “directed” as it sums weights of directed edges specifically from \( S \) to \( T \). The minimum cut problem on a graph is to find a cut that has the minimum cost among all cuts.

One of the fundamental results in combinatorial optimization is that the minimum s-t cut problem can be solved by finding a maximum flow from the source \( s \) to the sink \( t \). Loosely speaking, maximum flow is the maximum “amount of water” that can be sent from the source to the sink by interpreting graph edges as directed “pipes” with capacities equal to edge weights. The theorem of Ford and Fulkerson [146] states that a maximum flow from \( s \) to \( t \) saturates a set of edges in the graph, dividing the nodes into two disjoint parts \((S, T)\) corresponding to a minimum cut. Thus, min-cut and max-flow problems are equivalent. In fact, the maximum flow value is equal to the cost of the minimum cut. The close relationship between maximum flow and minimum cut problems is illustrated in figure 2.2 in the context of image segmentation. The max-flow displayed in figure 2.2a saturates the edges in the min-cut boundary in figure 2.2b. It turns out that max-flow and min-cut are dual problems, as explained in the appendix (section 2.5).

We can intuitively show how min-cut (or max-flow) on a graph may help with energy minimization over image labelings. Consider an example in figure 2.1. The graph corresponds to a \( 3 \times 3 \) image. Any s-t cut partitions the nodes into disjoint groups each containing exactly one terminal. Therefore, any cut corresponds to some assignment of pixels (nodes) to labels (terminals). If edge weights are appropriately set based on parameters of an energy, a minimum cost cut will correspond to a labeling with the minimum value of this energy.

2.1.3 Standard Algorithms in Combinatorial Optimization

An important fact in combinatorial optimization is that there are polynomial algorithms for min-cut/max-flow problems on directed weighted graphs with two terminals. Most well-known algorithms belong to one of the following groups: Ford–Fulkerson-style
“augmenting paths” [146, 126], the network simplex approach [170], and Goldberg–Tarjan-style “push-relabel” methods [171].

Standard augmenting path-based algorithms [146, 126] work by pushing flow along nonsaturated paths from the source to the sink until the maximum flow in the graph \( \mathcal{G} = (\hat{V}, \hat{E}, w) \) is reached. A typical augmenting path algorithm stores information about the distribution of the current \( s \to t \) flow \( f \) among the edges of \( \mathcal{G} \), using a residual graph \( \mathcal{G}_f \). The topology of \( \mathcal{G}_f \) is identical to \( \mathcal{G} \), but the capacity of an edge in \( \mathcal{G}_f \) reflects the residual capacity of the same edge in \( \mathcal{G} \), given the amount of flow already in the edge. At the initialization there is no flow from the source to the sink \( (f = 0) \) and edge capacities in the residual graph \( \mathcal{G}_0 \) are equal to the original capacities in \( \mathcal{G} \). At each new iteration the algorithm finds the shortest \( s \to t \) path along nonsaturated edges of the residual graph. If a path is found, then the algorithm augments it by pushing the maximum possible flow \( \Delta f \) that saturates at least one of the edges in the path. The residual capacities of edges in the path are reduced by \( \Delta f \) while the residual capacities of the reverse edges are increased by \( \Delta f \). Each augmentation increases the total flow from the source to the sink \( f = f + \Delta f \). The maximum flow is reached when any \( s \to t \) path crosses at least one saturated edge in the residual graph \( \mathcal{G}_f \).

The Dinic algorithm [126] uses a breadth-first search to find the shortest paths from \( s \) to \( t \) on the residual graph \( \mathcal{G}_f \). After all shortest paths of a fixed length \( k \) are saturated, the algorithm starts the breadth-first search for \( s \to t \) paths of length \( k + 1 \) from scratch. Note that the use of shortest paths is an important factor that improves theoretical running time complexities for algorithms based on augmenting paths. The worst case running time complexity for the Dinic algorithm is \( O(mn^2) \), where \( n \) is the number of nodes and \( m \) is the number of edges in the graph. In practice, the blocking flow approach of Dinic is known to outperform max-flow algorithms based on network simplex [170].
Push-relabel algorithms [171] use quite a different approach to the max-flow/min-cut problem. They do not maintain a valid flow during the operation; there are “active” nodes that have a positive “flow excess.” Instead, the algorithms maintain a labeling of nodes giving a lower bound estimate on the distance to the sink along nonsaturated edges. The algorithms attempt to “push” excess flows toward nodes with a smaller estimated distance to the sink. Typically, the “push” operation is applied to active nodes with the largest distance (label) or is based on a FIFO selection strategy. The distances (labels) progressively increase as edges are saturated by push operations. Undeliverable flows are eventually drained back to the source. We recommend our favorite textbook on basic graph theory and algorithms [101] for more details on push-relabel and augmenting path methods.

Note that the most interesting applications of graph cut to vision use directed N-D grids with locally connected nodes. It is also typical that a large portion of the nodes is connected to the terminals. Unfortunately, these conditions rule out many specialized min-cut/max-flow algorithms that are designed for some restricted classes of graphs. Examples of interesting but inapplicable methods include randomized techniques for dense undirected graphs [226] and methods for planar graphs assuming a small number of terminal connections [340, 188], among others.

### 2.1.4 The BK Algorithm

This section describes an algorithm developed as an attempt to improve empirical performance of standard augmenting path techniques on graphs in vision [68]. Normally (see section 2.1.3) augmenting path-based methods start a new breadth-first search for \( s \rightarrow t \) paths as soon as all paths of a given length are exhausted. In the context of graphs in computer vision, building a breadth-first search tree typically involves scanning the majority of image pixels. Practically speaking, it could be a very expensive operation if it has to be performed too often. Indeed, real-data experiments in vision confirm that rebuilding a search tree on graphs makes standard augmenting path techniques perform poorly in practice [68]. Several ideas were developed in [68] that improve the empirical performance of augmenting path techniques on sparse grid graphs in computer vision.

The BK (Boykov–Kolmogorov) algorithm [68] belongs to the group of algorithms based on augmenting paths. Similarly to Dinic [126], it builds search trees for detecting augmenting paths. In fact, BK builds two search trees, one from the source and the other from the sink. The other difference is that BK reuses these trees and never starts building them from scratch. The drawback of BK is that the augmenting paths found are not necessarily the shortest augmenting paths; thus the time complexity of the shortest augmenting path is no longer valid. The trivial upper bound on the number of augmentations for the BK algorithm is the cost of the minimum cut \( |C| \), which results in the worst case complexity \( O(mn^2|C|) \). Theoretically speaking, this is worse than the complexities of the standard algorithms discussed in section 2.1.3. However, experimental comparison conducted in
2.1 Algorithms for Min-Cut/Max-Flow Problems

Figure 2.3
Example of search trees $S$ (nodes with dense dots) and $T$ (nodes with sparse dots) at the end of the growth stage when a path (gray/yellow line) from the source $s$ to the sink $t$ is found. Active and passive nodes are labeled A and P, correspondingly. Free nodes are white.

[68, 11] shows that on many typical problem instances in vision (particularly for 2D cases) BK can significantly outperform standard max-flow algorithms.

Figure 2.3 illustrates BK’s basic ideas. The algorithm maintains two non-overlapping search trees $S$ and $T$ with roots at the source $s$ and the sink $t$, correspondingly. In tree $S$ all edges from each parent node to its children are non-saturated, while in tree $T$ edges from children to their parents are nonsaturated. The nodes that are not in $S$ or $T$ are “free.” We have

$S \subset V, \quad s \in S, \quad T \subset V, \quad t \in T, \quad S \cap T = \emptyset.$

The nodes in the search trees $S$ and $T$ can be either “active” or “passive”. The active nodes represent the outer border in each tree and the passive nodes are internal. The point is that active nodes allow trees to “grow” by acquiring new children (along nonsaturated edges) from a set of free nodes. The passive nodes cannot grow because they are completely blocked by other nodes from the same tree. It is also important that active nodes may come in contact with the nodes from the other tree. An augmenting path is found as soon as an active node in one of the trees detects a neighboring node that belongs to the other tree.

The algorithm iteratively repeats the following three stages:

• Growth stage: search trees $S$ and $T$ grow until they touch, giving an $s \rightarrow t$ path.
• Augmentation stage: the found path is augmented and search tree(s) break into forest(s).
• Adoption stage: trees $S$ and $T$ are restored.

At the growth stage the search trees expand. The active nodes explore adjacent nonsaturated edges and acquire new children from a set of free nodes. The newly acquired nodes become active members of the corresponding search trees. As soon as all neighbors of a given active node are explored, the active node becomes passive. The growth stage terminates if an active node encounters a neighboring node that belongs to the opposite tree. In this case BK detects a path from the source to the sink, as shown in figure 2.3.
The augmentation stage augments the path found at the growth stage. Since we push through the largest flow possible, some edge(s) in the path become(s) saturated. Thus, some of the nodes in the trees $S$ and $T$ may become “orphans,” that is, the edges linking them to their parents are no longer valid (they are saturated). In fact, the augmentation phase may split the search trees $S$ and $T$ into forests. The source $s$ and the sink $t$ are still roots of two of the trees and orphans form roots of all other trees.

The goal of the adoption stage is to restore the single-tree structure of sets $S$ and $T$ with roots in the source and the sink. At this stage we try to find a new valid parent for each orphan. A new parent should belong to the same set, $S$ or $T$, as the orphan. A parent should also be connected through a nonsaturated edge. If there is no qualifying parent, we remove the orphan from $S$ or $T$ and make it a free node. The algorithm also declares all its former children as orphans. The stage terminates when no orphans are left and, thus, the search tree structures of $S$ and $T$ are restored. Since some orphan nodes may become free, the adoption stage results in contraction of sets $S$ and $T$.

After the adoption stage is completed, the algorithm returns to the growth stage. The algorithm terminates when the search trees $S$ and $T$ cannot grow (no active nodes) and the trees are separated by saturated edges. This implies that a maximum flow is achieved. The corresponding minimum cut $C = \{S, T\}$ is defined as follows: nodes in the source search trees $S$ form subset $S$ and nodes in the sink tree $T$ form subset $T$. More details on implementing BK can be found in [68]. The code for the BK algorithm can be downloaded for research purposes from the authors’ Web pages.

### 2.1.5 Further Comments on Time and Memory Efficiency

The ability to compute globally optimal solutions for many large problems in computer vision is one of the main advantages of graph cut methods. Max-flow/min-cut algorithms outlined in the previous sections normally take only a few seconds to solve problems on graphs corresponding to typical 2D images [68, 11]. However, these algorithms may take several minutes to solve bigger 3D problems such as segmentation of medical volumetric data or multiview volumetric reconstruction.

Efficiency of max-flow/min-cut algorithms for graphs in computer vision is actively studied, and one should be aware that significant improvements are possible in specific situations. For example, one can use flow recycling [244] or cut recycling [221] techniques in dynamic applications (e.g., in video) where max-flow has to be computed for similar graphs corresponding to different time frames. Such methods are shown to work well when graphs and solutions change very little from one instance to the next. Methods for GPU-based acceleration of the push-relabel approach were also proposed [69]. Given fast advancements of the GPU technology, such methods may become very effective for grid graphs corresponding to 2D images.

Limited memory resources could be a serious problem for large N-D problems, as standard max-flow/min-cut algorithms practically do not work as soon as the whole graph does
not fit into available RAM. This issue was addressed in [120], where a scalable version of the push-relabel method was developed specifically for large regular N-D graphs (grids or complexes).

Note that the speed and memory efficiency of different max-flow/min-cut algorithms can be tested on problems in computer vision using a database of graphs (in the standard DIMACS format) that recently become available. The posted graphs are regular grids or complexes corresponding to specific examples in segmentation, stereo, multiview reconstruction, and other applications.

2.2 Max-flow Algorithm as an Energy Minimization tool

Section 2.1 described the min-cut/max-flow problem and several max-flow algorithms. As already mentioned, they have a close relationship with the problem of minimizing pseudo-Boolean functions of the form (2.1). To see this, consider an undirected graph \((V, E)\) and the corresponding directed weighted graph \((\hat{V}, \hat{E}, w)\) defined in section 2.1.1. Any s-t cut \((S, T)\) can be viewed as a binary labeling \(x\) of the set of nodes \(V\) defined as follows: \(x_i = 0\) if \(i \in S\), and \(x_i = 1\) if \(i \in T\) for node \(i \in V\). The energy of labeling \(x\) (i.e., the cost of the corresponding cut \((S, T)\)) equals

\[
E(x) = \sum_{i \in V} (w_{si} x_i + w_{ti} \bar{x}_i) + \sum_{(i,j) \in E} (w_{ij} x_i x_j + w_{ji} \bar{x}_i \bar{x}_j). \tag{2.4}
\]

There is a 1:1 correspondence between binary labelings and cuts; therefore, computing a minimum cut in \((\hat{V}, \hat{E}, w)\) will yield a global minimum of function (2.4). Note that standard max-flow/min-cut algorithms reviewed in sections 2.1.3 and 2.1.4 work only for graphs with nonnegative weights \(w \geq 0\). Thus, equation (2.4) implies that such algorithms can minimize quadratic pseudo-Boolean functions (2.1) with coefficients \((\theta_i; 0), (\theta_i; 1), (\theta_{ij}; 00), (\theta_{ij}; 01), (\theta_{ij}; 10), (\theta_{ij}; 11)\) satisfying the following conditions:

\[
\begin{align*}
\theta_i; 0 & \geq 0 & \quad \theta_i; 1 & \geq 0 \quad \text{(2.5a)} \\
\theta_{ij}; 00 &= 0 & \quad \theta_{ij}; 01 & \geq 0 & \quad \theta_{ij}; 10 & \geq 0 & \quad \theta_{ij}; 11 &= 0 \quad \text{(2.5b)}
\end{align*}
\]

for all nodes \(i \in V\) and edges \((i, j) \in E\).

Can we use a max-flow algorithm for a more general class of quadratic pseudo-Boolean functions? In the next section we define operations on coefficients \(\theta\) that do not change the energy \(E(x)\); such operations are called reparameterizations. We will then explore which functions can be reparameterized to satisfy (2.5), arriving at the class of submodular...
functions. In section 2.2.2 we will see that any submodular QPB (quadratic pseudo-Boolean) function can be transformed to (2.5), and thus can be minimized in polynomial time via a max-flow algorithm.

Unfortunately, this transformation does not work for nonsubmodular functions, which should not be surprising, given that minimizing such functions is an NP-hard problem (it includes, e.g., the maxcut problem, which is NP-hard [156]). Still, the max-flow algorithm can be quite useful for certain nonsubmodular functions, (e.g., it can identify a part of an optimal labeling). We will review some known results in section 2.3.

2.2.1 Reparameterization and Normal Form

Recall that energy (2.1) depends on vector $\theta$, which is a concatenation of all coefficients $\theta_{\text{const}}, \theta_{i:a}$, and $\theta_{ij:ab}$. To emphasize this dependence, we will often write the energy defined by $\theta$ as $E(x \mid \theta)$ instead of $E(x)$.

Given some node $i \in V$ and some real number $\delta \in \mathbb{R}$, consider an operation transforming vector $\theta$ as follows:

\[
\begin{align*}
\theta_{i:0} &:= \theta_{i:0} - \delta & \theta_{i:1} &:= \theta_{i:1} - \delta & \theta_{\text{const}} &:= \theta_{\text{const}} + \delta
\end{align*}
\]  

(2.6)

where “:=” denotes the assignment operator as used in programming languages. It is easy to see that this transformation does not change the function $E(x \mid \theta)$ and the cost of any labeling $x$ stays the same. This follows from the fact that $x_i + \overline{x}_i = 1$. This motivates the following definition:

**Definition 2.1** Vector $\theta'$ is called a reparameterization of vector $\theta$ if they define the same energy function, that is, $E(x \mid \theta') = E(x \mid \theta)$ for any labeling $x.$ In this case we may also write $\theta' \sim \theta$.

Consider an edge $(i, j) \in E$. Identities $\overline{x}_j = \overline{x}_j (x_i + \overline{x}_i)$ and $x_j = x_j (x_i + \overline{x}_i)$ yield two more reparameterization operations:

\[
\begin{align*}
\theta_{ij:00} &:= \theta_{ij:00} - \delta & \theta_{ij:10} &:= \theta_{ij:10} - \delta & \theta_{j:0} &:= \theta_{j:0} + \delta
\end{align*}
\]  

(2.7a)

\[
\begin{align*}
\theta_{ij:01} &:= \theta_{ij:01} - \delta & \theta_{ij:11} &:= \theta_{ij:11} - \delta & \theta_{j:1} &:= \theta_{j:1} + \delta.
\end{align*}
\]  

(2.7b)

Similarly, identities $\overline{x}_i = \overline{x}_i (x_j + \overline{x}_j)$ and $x_i = x_i (x_j + \overline{x}_j)$ give

\[
\begin{align*}
\theta_{ij:00} &:= \theta_{ij:00} - \delta & \theta_{ij:01} &:= \theta_{ij:01} - \delta & \theta_{i:0} &:= \theta_{i:0} + \delta
\end{align*}
\]  

(2.7c)

\[
\begin{align*}
\theta_{ij:10} &:= \theta_{ij:10} - \delta & \theta_{ij:11} &:= \theta_{ij:11} - \delta & \theta_{i:1} &:= \theta_{i:1} + \delta.
\end{align*}
\]  

(2.7d)

Operation 2.7a is illustrated in figure 2.4b. It can be shown that any possible reparameterization can be obtained as a combination of operations (2.6 and 2.7), assuming that graph $(V, E)$ is connected (e.g., [520]).
2.2 Max-flow Algorithm as an Energy Minimization tool

Submodular term

Supermodular term

Figure 2.4
(a) Convention for displaying parameters \( \theta_{i:a}, \theta_{i:ab}, \theta_{j:b} \).
(b) Example of a reparameterization operation (equation 2.7a).
(c) Normal form. Dotted lines denote links with zero cost. The first term is submodular, the second is supermodular. Unary parameters must satisfy \( \min\{\theta_{i:0}, \theta_{i:1}\} = \min\{\theta_{j:0}, \theta_{j:1}\} = \min\{\theta_{k:0}, \theta_{k:1}\} = 0 \).

Definition 2.2 Vector \( \theta \) is in a normal form if each node \( i \in V \) satisfies

\[
\min\{\theta_{i:0}, \theta_{i:1}\} = 0
\]  

(2.8)

and each edge \( (i, j) \in E \) satisfies either (2.9a) or (2.9b) below:

\[
\begin{align*}
\theta_{ij:00} & = 0, & \theta_{ij:01} & \geq 0, & \theta_{ij:10} & \geq 0, & \theta_{ij:11} & = 0 \\
\theta_{ij:00} & \geq 0, & \theta_{ij:01} & = 0, & \theta_{ij:10} & = 0, & \theta_{ij:11} & \geq 0.
\end{align*}
\]  

(2.9a)

(2.9b)

Figure 2.4c illustrates conditions (2.9a) and (2.9b) on edges \( (i, j) \) and \( (j, k) \), correspondingly. Note that (2.9a) agrees with (2.5b).

It is not difficult to verify that any quadratic pseudo-Boolean function \( E(x | \theta) \) can be reparameterized to a normal form in linear time. For example, this can be done by the following simple algorithm:

1. For each edge \( (i, j) \in E \) do the following:
   • Make \( \theta_{ij:ab} \) nonnegative: compute \( \delta = \min_{a, b \in \{0, 1\}} \theta_{ij:ab} \) and set
     \[
     \theta_{ij:ab} := \theta_{ij:ab} - \delta \quad \forall \, a, b \in \{0, 1\}, \quad \theta_{\text{const}} := \theta_{\text{const}} + \delta.
     \]
   • For each label \( b \in \{0, 1\} \) compute \( \delta = \min\{\theta_{ij:0b}, \theta_{ij:1b}\} \) and set
     \[
     \theta_{ij:0b} := \theta_{ij:0b} - \delta, \quad \theta_{ij:1b} := \theta_{ij:1b} - \delta, \quad \theta_{j:b} := \theta_{j:b} + \delta.
     \]
   • For each label \( a \in \{0, 1\} \) compute \( \delta = \min\{\theta_{ij:a0}, \theta_{ij:a1}\} \) and set
     \[
     \theta_{ij:a0} := \theta_{ij:a0} - \delta, \quad \theta_{ij:a1} := \theta_{ij:a1} - \delta, \quad \theta_{i:a} := \theta_{i:a} + \delta.
     \]

2. For each node \( i \) compute \( \delta = \min\{\theta_{i:0}, \theta_{i:1}\} \) and set
   \[
   \theta_{i:0} := \theta_{i:0} - \delta, \quad \theta_{i:1} := \theta_{i:1} - \delta, \quad \theta_{\text{const}} := \theta_{\text{const}} + \delta.
   \]
The first step of this algorithm performs a fixed number of operations for each edge, and the second step performs a fixed number of operations for each node, giving linear overall complexity $O(|\mathcal{V}| + |\mathcal{E}|)$.

In general, the normal form is not unique for vector $\theta$. For example, in the appendix it is shown that each augmentation of a standard max-flow algorithm can be interpreted as a reparameterization of energy $E(x \mid \theta)$ from one normal form to another. The set of all reparameterizations of $\theta$ in a normal form will be denoted by $\Omega(\theta)$:

$$\Omega(\theta) = \{ \theta' \sim \theta \mid \theta' \text{ is in normal form} \}.$$

### 2.2.2 Submodularity

It is easy to check that reparameterization operations (2.7) for edge $(i, j)$ preserve the quantity

$$\Delta_{ij} = (\theta_{ij;00} + \theta_{ij;11}) - (\theta_{ij;01} + \theta_{ij;10}).$$

Thus, $\Delta_{ij}$ is an invariant that can be used to classify each edge $(i, j)$.

**Definition 2.3**

(a) Edge $(i, j)$ is called submodular with respect to given energy function $E(x \mid \theta)$ if $\Delta_{ij} \leq 0$. It is called supermodular if $\Delta_{ij} \geq 0$.

(b) A quadratic pseudo-Boolean function $E(x \mid \theta)$ is called submodular if all its edges $(i, j)$ are submodular, that is, they satisfy

$$\theta_{ij;00} + \theta_{ij;11} \leq \theta_{ij;01} + \theta_{ij;10}.$$

We have already seen a definition of submodularity for general pseudo-Boolean functions in chapter 1 (1.59). It is not difficult to show that for quadratic pseudo-Boolean functions the two definitions are equivalent. We leave this proof as an exercise.

Equation 2.11 encodes some notion of smoothness. It says that the combined cost of homogeneous labelings $(0, 0)$ and $(1, 1)$ is smaller than the combined cost of discontinuous labelings $(0, 1)$ and $(1, 0)$. The smoothness assumption is natural in vision problems where nearby points are likely to have similar labels (object category, disparity, etc.).

Note that for energy $E(x \mid \theta)$ in a normal form, condition (2.9a) describes submodular edges and condition (2.9b) describes supermodular edges (see also fig.2.4(c)).

Now we can return to the question of what energy functions $E(x \mid \theta)$ can be reparameterized to the form of equations (2.5). Any energy can be transformed to a normal form by a simple reparameterization algorithm described in section 2.2.1. Condition (2.8) will satisfy (2.5a). Since submodularity is preserved under reparameterization, all submodular edges will satisfy (2.9a) and, thus, (2.5b). This proves the following.
Theorem 2.1  Global minima $\hat{x} = \arg\min_x E(x \mid \theta)$ of any submodular quadratic pseudo-Boolean function can be obtained in polynomial time, using the following steps:

1. Reparameterize $\theta$ to a normal form as described in section 2.2.1.
2. Construct the directed weighted graph $\hat{G} = (\hat{V}, \hat{E}, w)$ as described in section 2.1.1 with the following nonnegative arc capacities:
   
   $w_{xi} = \theta_{i;1}, \quad w_{it} = \theta_{i;0}, \quad w_{ij} = \theta_{ij;01}, \quad w_{ji} = \theta_{ji;10}.$

3. Compute a minimum $s$-$t$ cut in $\hat{G}$ and the corresponding labeling $x$.

Note that supermodularity is also preserved under reparameterization. After conversion to a normal form any (strictly) supermodular edge will satisfy (2.9b), which is inconsistent with (2.5b). Therefore, nonsupermodular functions with one or more supermodular edges cannot be converted to (2.5) and, in general, they cannot be minimized by standard max-flow algorithms. Optimization of nonsubmodular functions is known to be an NP-hard problem. However, section 2.3 describes one approach that may work for some cases of non-submodular functions.

2.3 Minimizing Nonsubmodular Functions

Let us now consider the case when not all edges for function $E(x)$ satisfy the submodularity condition (2.11). Minimizing such a function is an NP-hard problem, so there is little hope that there exists a polynomial time algorithm for solving arbitrary instances. It does not mean, however, that all instances that occur in practice are equally hard. This section describes a linear programming relaxation approach that can solve many of the instances that occur in computer vision applications, as can be seen in several other chapters of the book.

Relaxation is a general technique applicable to many optimization problems. It can be described as follows. Suppose that we want to minimize function $E(x)$ over a set $X \subset \mathbb{Z}^n$ containing a finite number of integer-valued points. First, we extend function $E$ to a larger domain $\hat{X}$, $X \subset \hat{X}$ so that $\hat{X}$ is a convex subset of $\mathbb{R}^n$. Assuming that $E$ is a convex function over $\hat{X}$, one can usually compute a global minimizer $\hat{x}$ of function $E$ over $\hat{X}$, using one of many efficient methods for convex optimization problems. (More information on relaxation techniques can be found in chapter 16.)

For some “easy” instances it may happen that a minimizer $\hat{x}$ is integer-valued and lies in the original domain $X$; then we know that we have solved the original problem. In general, however, the vector $\hat{x}$ may have fractional components. In that case there are several possibilities. We may try to round $\hat{x}$ to an integer-valued solution so that the objective function $E$ does not increase too much. (This scheme is used for a large number of approximation algorithms in combinatorial optimization.) Another option is to use the minimum value
\[
\min_{x \in \hat{X}} E(x) \text{ as a lower bound on the original optimization problem in a branch-and-bound framework.}
\]

This section reviews the LP relaxation of energy (2.1) introduced earlier, in section 1.9 of chapter 1:

Minimize \( \theta_{\text{const}} + \sum_{\substack{i \in V, \\ a \in \{0,1\}}} \theta_{i:a} y_{i:a} + \sum_{\substack{(i,j) \in E, \\ a,b \in \{0,1\}}} \theta_{ij:ab} y_{ij:ab} \) \hspace{1cm} (2.12a)

subject to \( y_{ij:0b} + y_{ij:1b} = y_{j:b} \) \hspace{1cm} \forall (i,j) \in E, b \in \{0,1\}, \hspace{1cm} (2.12b)

\( y_{ij:a0} + y_{ij:a1} = y_{i:a} \) \hspace{1cm} \forall (i,j) \in E, a \in \{0,1\}, \hspace{1cm} (2.12c)

\( y_{i:0} + y_{i:1} = 1 \) \hspace{1cm} \forall i \in V, \hspace{1cm} (2.12d)

\( y_{i:a}, y_{ij:ab} \in [0,1] \) \hspace{1cm} \forall i \in V, (i,j) \in E, a,b \in \{0,1\}. \hspace{1cm} (2.12e)

Note that forcing variables \( y_{i:a}, y_{ij:ab} \) to be integral would make (2.12) equivalent to the minimization problem (2.1); \( y_{i:a} \) and \( y_{ij:ab} \) would be the indicator variables of events \( x_i = a \) and \( (x_i, x_j) = (a,b) \), respectively. Thus, (2.12) is indeed a relaxation of (2.1). It is known as the roof duality relaxation [179, 61].

Equation (2.12) is a linear program. (Interestingly, it can be shown to be the LP dual of problem (DUAL) formulated in the appendix to this chapter. Thus, there is a strong duality between (2.17) and LP relaxation (2.12), but there may be a “duality gap” between (2.17) and the original integer problem (2.1) in a general nonsubmodular case.) It can be shown [179] that the extreme points of this LP are half-integral. That is, an optimal solution \( \hat{y} \) may have components \( \hat{y}_{i:a}, \hat{y}_{ij:ab} \) belonging to \( \{0,1,\frac{1}{2}\} \). Two important questions arise:

- Does \( \hat{y} \) give any information about minimizers of function (2.1)? This is discussed in section 2.3.1.
- How can \( \hat{y} \) be computed? Since (2.12) is a linear program, one could use a number of generic LP solvers. However, there exist specialized combinatorial algorithms for solving (2.12). The method in [62] based on the max-flow algorithm is perhaps the most efficient; it is reviewed in section 2.3.2. We will refer to it as the BHS algorithm.

### 2.3.1 Properties of Roof Duality Relaxation

Let \( \hat{y} \) be a half-integral optimal solution of (2.12). It is convenient to define a partial labeling \( x \) as follows: \( x_i = \hat{y}_{i:1} \) if \( \hat{y}_{i:1} \in \{0,1\} \) and \( x_i = \emptyset \) if \( \hat{y}_{i:1} = \frac{1}{2} \). In the former case we say node \( i \) is labeled; otherwise it is unlabeled.

An important property of roof duality relaxation is that \( x \) gives a part of an optimal solution. In other words, function \( E \) has a global minimum \( x^* \) such that \( x^*_i = x_i \) for all labeled nodes \( i \). This property is known as persistency or partial optimality [179, 61, 261].
Clearly, the usefulness of roof duality relaxation depends on how many nodes are labeled and this depends heavily on the application. If the number of nonsubmodular terms is small and they are relatively weak compared with unary terms, then we can expect most nodes to be labeled. In other situations all nodes can remain unlabeled. Refer to [403] and chapter 18 for some computational experiments in computer vision.

An important question is what to do with remaining unlabeled nodes. If the number of such nodes is small or the remaining graph has low tree width, then one could use a junction tree algorithm to compute a global minimum. Another option is to use the PROBE technique [63]. The idea is to fix unlabeled nodes to a particular value (0 or 1) and apply roof duality relaxation to the modified problem. After this operation more nodes may become labeled, giving further information about minimizers of function $E$. This information is used to simplify function $E$, and the process is repeated until we cannot infer any new constraints on minimizers of $E$. For certain functions the PROBE procedure labels many more nodes compared with the basic roof duality approach [63, 403].

In computer vision applications QPB functions are often used inside iterative move-making algorithms, such as expansion moves (chapter 3) or fusion moves (chapter 18). In that case the roof duality approach can be used as follows. Suppose that the current configuration is represented by a binary labeling $z^o$, and solving the LP relaxation gives partial labeling $x$. Let us replace $z^o$ with the following labeling $z$: if node $i$ is labeled, then $z_i = x_i$, otherwise $z_i = z^o_i$. In such an iterative optimization the energy can never increase. This follows from the autarky property that says $E(z) \leq E(z^o)$ [179, 61].

### 2.3.2 Solving Roof Duality Relaxation via Max-flow: The BHS Algorithm

The LP relaxation (2.12) can be solved in many different ways, including generic LP solvers. We review the method in [62], which is perhaps the most efficient, and we call this the BHS algorithm. On the high level, it can be described as follows. The original energy (2.1) is relaxed to a symmetric submodular function using binary indicator variables $x_i; 1$ and $x_i; 0$, introduced for each variable $x_i$ and its negation $\overline{x}_i = 1 - x_i$, respectively. This submodular formulation can be solved by max-flow algorithms. The corresponding result can also be shown to solve LP (2.12).

Below we provide more details. The first step is to reparameterize the energy $E(x | \theta)$ into a normal form, as described in section 2.2.1. Then, submodular edges will have the form (2.9a) and supermodular edges will have the form (2.9b). The second step is to construct

---

3. The code can be found online on V. Kolmogorov’s Web page. This includes the BHS algorithm, the PROBE procedure, and the heuristic IMPROVE technique (see [403]).

4. BHS is an abbreviation of the inventors’ names. Sometimes it has also been called the QPBO algorithm in the computer vision literature, as in [403]. This is, however, misleading; the optimization problem is already called QPBO (chapter 1).
a new energy function \( \tilde{E}(\{x_i; 1\}, \{x_i; 0\}) \) by transforming terms of the original function (2.1) as follows:

\[
\begin{align*}
\theta_i;0 x_i & \quad \mapsto \quad \frac{\theta_i;0}{2} [x_i;1 + x_i;0] \\
\theta_i;1 x_i & \quad \mapsto \quad \frac{\theta_i;1}{2} [x_i;1 + \overline{x}_i;0] \\
\theta_{ij};01 x_i x_j & \quad \mapsto \quad \frac{\theta_{ij};01}{2} [x_i;1 x_j;1 + x_i;0 \overline{x}_j;0] \\
\theta_{ij};10 x_i x_j & \quad \mapsto \quad \frac{\theta_{ij};10}{2} [x_i;1 \overline{x}_j;1 + x_i;0 x_j;0] \\
\theta_{ij};00 x_i x_j & \quad \mapsto \quad \frac{\theta_{ij};00}{2} [x_i;0 \overline{x}_j;1 + \overline{x}_i;1 x_j;0] \\
\theta_{ij};11 x_i x_j & \quad \mapsto \quad \frac{\theta_{ij};11}{2} [x_i;1 \overline{x}_j;0 + \overline{x}_i;0 x_j;1].
\end{align*}
\]

(2.13a) (2.13b) (2.13c) (2.13d) (2.13e) (2.13f)

The constant term \( \theta_{\text{const}} \) remains unmodified.

The constructed function \( \tilde{E} \) has several important properties. First, it is equivalent to the original function if we impose the constraint \( x_i;0 = 1 - x_i;1 \) for all nodes \( i \):

\[
\begin{align*}
\tilde{E}(x, \overline{x}) &= E(x) \quad \forall x.
\end{align*}
\]

(2.14)

Second, function \( \tilde{E} \) is submodular. (Note that all pairwise terms are of the form \( c \cdot x_i; a \overline{x}_j; b \) where \( c \geq 0 \) and \( x_i; a, x_j; b \) are binary variables.) This means that we can minimize this function by computing a maximum flow in an appropriately constructed graph (see section 2.2). This graph will have twice as many nodes and edges compared with the graph needed for minimizing submodular functions \( E(x) \).

After computing a global minimum \( ((x_i;1), (x_i;0)) \) of function \( \tilde{E} \) we can easily obtain a solution \( \hat{y} \) of the relaxation (2.12). The unary components for node \( i \in V \) are given by

\[
\begin{align*}
\hat{y}_i;1 &= \frac{1}{2} [x_i;1 + \overline{x}_i;0] \\
\hat{y}_i;0 &= \frac{1}{2} [x_i;0 + \overline{x}_i;1].
\end{align*}
\]

(2.15a) (2.15b)

Clearly, we have \( \hat{y}_i;1 + \hat{y}_i;0 = 1 \) and \( \hat{y}_i;1, \hat{y}_i;0 \in \{0, 1, \frac{1}{2}\} \). It also is not difficult to derive pairwise components \( \hat{y}_{ij;ab} \): we just need to minimize (2.12) over \( \{\hat{y}_{ij;ab}\} \) with fixed unary components \( \{\hat{y}_{i;a}\} \). We omit the formulas because they are rarely used in practice.

A complete proof that the algorithm above indeed solves problem (2.12) is a bit involved, and we do not give it here. The original proof can be found in [61]. Below we provide a sketch of an alternative proof that follows the general approach in [195]. The idea is to establish
some correspondence between LP relaxation (2.12) of the original nonsubmodular energy $E$ and the same type of relaxation for submodular energy $\tilde{E}$. To be concise, we will refer to these two relaxation problems as $LP$ and $\tilde{LP}$. Assuming that $\mathcal{F}(LP)$ and $\mathcal{F}(\tilde{LP})$ are the sets of feasible solutions of the two problems, it is possible to define linear mappings $\varphi : \mathcal{F}(LP) \rightarrow \mathcal{F}(\tilde{LP})$ and $\psi : \mathcal{F}(\tilde{LP}) \rightarrow \mathcal{F}(LP)$, which preserve the cost of the solutions. This implies that $LP$ and $\tilde{LP}$ have the same minimum value, and any solution of one problem yields an optimal solution for the other one. Relaxation $\tilde{LP}$ corresponds to a submodular energy, and therefore it can be solved via a max-flow algorithm. Applying mapping $\psi$ to the optimal solution of $\tilde{LP}$ yields formulas (2.15).

2.4 Conclusions

This chapter described standard optimization techniques for binary pairwise models. A large number of problems in the following chapters are represented using such quadratic pseudo-Boolean functions. Equation (2.1) is a different representation of energy convenient for binary variables $x$. This representation is common in combinatorial optimization literature, where optimization of pseudo-Boolean functions has been actively studied for more than forty years.

This chapter studied an important class of pseudo-Boolean functions whose minimization can be reduced to min-cut/max-flow problems on graphs. Such functions are characterized by a submodularity condition. We also reviewed several classical polynomial complexity algorithms (augmenting paths [146, 126] and push-relabel [171]) for solving min-cut/max-flow problems. Efficient versions of such algorithms were also developed specifically for sparse grids common in computer vision [68, 120]. This chapter reviewed the BK algorithm [68] that is widely used in imaging.

Section 2.3 concluded this chapter by describing more general graph cut techniques that can be applied to nonsubmodular binary models. Such general quadratic pseudo-Boolean optimization (QPBO) methods are based on roof duality relaxation of the integer programming problem. They are not guaranteed to find the global minima solution, but they can find partial solutions.

2.5 Appendix

2.5.1 Max-flow as a Reparameterization

We showed how to minimize any submodular QPB function using a max-flow algorithm. At this point the reader may ask why we solve a maximization problem (computing a flow of maximum value) for minimizing a function. As was mentioned in section 2.2, the maximization and minimization problems are dual to one another. In this section we will illustrate this duality using the notion of reparameterization.
First, let us formulate the dual problem in the context of energy minimization. Without loss of generality, we will assume that energy \( E(x \mid \theta) \) was already converted to a normal form (2.8 and 2.9), as described in section 2.2.1. It is easy to see that for any vector \( \theta \) with nonnegative components \( \{\theta_i; a\}, \{\theta_{ij}; ab\} \), the constant term \( \theta_{\text{const}} \) is a lower bound on the function \( E(x \mid \theta) \) defined by (2.1), that is,

\[
\theta_{\text{const}} \leq \min_x E(x \mid \theta). \tag{2.16}
\]

In order to obtain the tightest possible bound, one can solve the following maximization problem:

\((\text{DUAL})\) Given \( \theta \), find its reparameterization \( \theta^* \) with nonnegative components \( \{\theta^*_i; a\}, \{\theta^*_{ij}; ab\} \) such that the lower bound \( \theta^*_{\text{const}} \) is maximized.

It is easy to check that the solution to (DUAL) can be found among normal form reparameterizations of \( \theta \), that is, (DUAL) is equivalent to

\[
\max_{\theta' \in \Omega(\theta)} \theta'_{\text{const}}. \tag{2.17}
\]

It is not difficult to show that problems (DUAL) and (2.17) correspond to some linear programs.\(^5\) Also note that inequality (2.16) and problems (DUAL) and (2.17) make sense for both submodular and nonsubmodular functions. In any case, \( \theta_{\text{const}} \) is a lower bound on \( \min_x E(x \mid \theta) \) as long as vector \( \theta \) is nonnegative (e.g., in normal form).

When the energy function \( E(x \mid \theta) \) is submodular, (2.17) is the exact maximization problem solved by max-flow algorithms on graph \( \hat{G} \) described in theorem 2.1. For example, consider an augmenting path-style algorithm from section 2.1.3. Pushing flow through an arc changes residual capacities of this arc and its reverse arc, thus changing vector \( \theta \).\(^6\)

If we send \( \delta \) units of flow from the source to the sink via arcs \( (s \rightarrow i_1), (i_1 \rightarrow i_2), \ldots, (i_{k-1} \rightarrow i_k), (i_k \rightarrow t) \), then the corresponding transformation of vector \( \theta \) is

\[
\begin{align*}
\theta_{i_1:1} &:= \theta_{i_1:1} - \delta \\
\theta_{i_1i_2:01} &:= \theta_{i_1i_2:01} - \delta \\
\theta_{i_1i_2:10} &:= \theta_{i_1i_2:10} + \delta \\
\vdots &\end{align*}
\begin{align*}
\theta_{i_{k-1}i_k:01} &:= \theta_{i_{k-1}i_k:01} - \delta \\
\theta_{i_{k-1}i_k:10} &:= \theta_{i_{k-1}i_k:10} + \delta \\
\theta_{i_k:0} &:= \theta_{i_k:0} - \delta \\
\theta_{\text{const}} &:= \theta_{\text{const}} + \delta.
\end{align*}
\tag{2.18}
\]

\(^5\) The definition of reparameterization involves an exponential number of linear constraints. But the fact that any reparameterization \( \theta' \sim \theta \) can be expressed via operations (2.6 and 2.7) allows polynomial size LP formulations for (DUAL) and (2.17).

\(^6\) Recall that the reduction given in theorem 2.1 associates each arc of graph \( \hat{G} \) with one component of vector \( \theta \). Thus, we can define a 1:1 mapping between residual capacities of graph \( \hat{G} \) and components of vector \( \theta \).
It can be checked that this transformation of $\theta$ is indeed a reparameterization obtained by combining operations (2.6 and 2.7) for the sequence of edges above. Furthermore, it keeps vector $\theta$ in a normal form and increases the lower bound by $\delta$. Thus, each augmentation greedily improves the objective function of the maximization problem (2.17). This relationship between reparameterization and augmenting path flows has been used to design dynamic algorithms for energy minimization [66, 221, 244].

Upon termination the max-flow algorithm yields a minimum $s$-$t$ cut $(S, T)$ corresponding to some labeling $\hat{x}$ and a maximum flow corresponding to some reparameterization $\hat{\theta}$. Using the property that all arcs from $S$ to $T$ have zero residual capacity, one can check that $\hat{\theta}_{i;\hat{x}_i} = 0$ for all nodes $i$ and $\hat{\theta}_{ij;\hat{x}_i\hat{x}_j} = 0$ for all edges $(i, j)$. Therefore, from equation (2.1’), we get

$$E(\hat{x} \mid \hat{\theta}) = \hat{\theta}_{\text{const}} + \sum_{i \in V} \hat{\theta}_{i;\hat{x}_i} + \sum_{(i,j) \in E} \hat{\theta}_{ij;\hat{x}_i\hat{x}_j} = \hat{\theta}_{\text{const}},$$

that is, lower bound $\hat{\theta}_{\text{const}}$ in (2.16) equals the cost of labeling $\hat{x}$. This confirms the optimality of $\hat{x}$ and $\hat{\theta}$. Moreover, (2.19) implies that for submodular energies $E(x \mid \theta)$ there is no duality gap, that is,

$$\max_{\theta' \in \Omega(\theta)} \theta'_{\text{const}} = \min_x E(x \mid \theta),$$

which is also known as a strong duality relationship.

Note that inequality (2.16) holds for arbitrary quadratic pseudo-Boolean functions $E(x \mid \theta)$ in a normal form including any combination of submodular and supermodular terms. Presence of supermodular terms, however, may result in the optimal lower bound in the problem (DUAL) being smaller than the minimum of $E(x \mid \theta)$. In other words, we may have a weak duality relationship $\max_{\theta' \in \Omega(\theta)} \theta'_{\text{const}} < \min_x E(x \mid \theta)$, instead of the strong duality (2.20). In the next chapter we will study some optimization algorithms for general non-submodular quadratic pseudo-Boolean functions that may include supermodular terms.