Chapter 11

Undirected Models for Images

In this chapter, we once more consider models that associate a label with each pixel of a large image. We assume that the data at the pixel provides only very ambiguous information about the associated label. However, certain spatial configurations of labels are known to be more common than others and we aim to exploit this knowledge to resolve the ambiguity.

In chapter 10 the prior knowledge about plausible label configurations took the form of a directed model in which we specified the conditional probability of a label given its topological parents. In this chapter, we describe the relative preference for configurations of labels with a pairwise *Markov random field* or MRF. This is an undirected model in which the interactions between labels are symmetric and there is no notion of parent or child.

The properties of the MRF necessitate quite different approaches to learning and inference than those used for directed models, and these changes have important implications: maximum a posteriori (MAP) inference becomes tractable in many circumstance, but learning becomes more problematic.

11.1 Denoising

Before we discuss undirected models, we introduce a representative application. In *image denoising* (figure 11.1) we observe a corrupted image and aim to recover the original. We will consider a noise process where a certain proportion of pixels have been randomly changed to another value according to a uniform distribution.

More precisely, the observed image $\mathbf{x} = \{x_1, x_2 \dots x_N\}$ is assumed to consist of discrete variables where the different possible values (labels) represents different intensities. Our goal is to recover the original uncorrupted image $\mathbf{y} = \{y_1, y_2 \dots y_N\}$ which also consists of discrete variables representing the intensity. We will initially restrict our discussion to generative models and compute the posterior probability over the unknown world state $\mathbf{y}$ using Bayes' rule

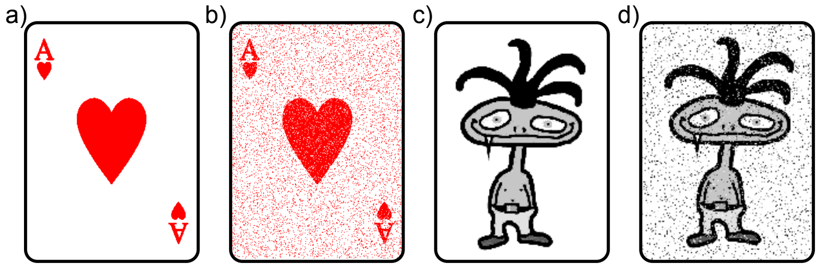

Figure 11.1 Image denoising. a) Original binary image. b) Observed image created by randomly flipping the polarity of a fixed proportion of pixels. Our goal is to recover the original image from the corrupted one. c) Original grayscale image. d) Observed corrupted image is created by setting a certain proportion of the pixels to values drawn from a uniform distribution. Once more, we aim to recover the original image.

$$Pr(y_{1\ldots N}|x_{1\ldots N}) = \frac{\prod_{n=1}^{N} Pr(x_n|y_n)Pr(y_{1\ldots N})}{Pr(x_{1\ldots N})}, \tag{11.1}$$

where we have assumed that the data x_n at pixel n is conditionally independent of all the other true pixel values $y_{1\ldots N\backslash n}$ given the associated true pixel value y_n. We first consider denoising binary images in which the noise process flips the pixel polarity with probability ρ so that

$$\begin{aligned} Pr(x_n|y_n = 0) &= \text{Bern}_{x_n}[\rho] \\ Pr(x_n|y_n = 1) &= \text{Bern}_{x_n}[1-\rho] \end{aligned} \tag{11.2}$$

We subsequently consider gray level denoising where the observed pixel is replaced with probability ρ with a draw from a uniform distribution .

The focus of the remaining part of this chapter is on the prior $Pr(y_{1\ldots N})$, which we will choose to be a Markov Random field. This is a special case of a more general class of distribution called undirected models.

11.2 Undirected Models

Undirected models describe the probability over a set of variables $y_{1\ldots N}$ as a product of *potential functions* $\phi[\bullet]$ over the variables so that

$$Pr(y_{1\ldots N}) = \frac{1}{Z}\prod_{c=1}^{C}\phi_c[y_{1\ldots N}] \tag{11.3}$$

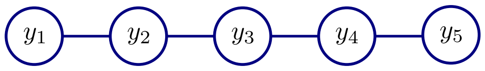

Figure 11.2 Graphical model for worked MRF example. The variables form a chain in which only neighbours are connected. However, the links in the chain have no particular direction: this is an *undirected model.*

where the potential function $\phi_c[y_{1\ldots N}]$ always returns a positive number. Since the probability increases when $\phi_c[y_{1\ldots K}]$ increases, the function modulates the tendency for the system to take values $y_{1\ldots N}$. The probability is greatest where all of the functions $\phi_{1\ldots C}$ return high values. The term Z is known as the *partition function* and normalizes the product of these positive functions so that the total probability is one:

$$Z = \sum_{y_1}\sum_{y_2}\cdots\sum_{y_N}\prod_{c=1}^{C}\phi_c[y_{1\ldots N}] \tag{11.4}$$

We can alternatively write equation 11.3 as

$$Pr(y_{1\ldots N}) = \frac{1}{Z}\exp\left[-\sum_{c=1}^{C}\psi_c[y_{1\ldots N}]\right] \tag{11.5}$$

where $\phi_c[y_{1\ldots N}] = -\log[\psi_c[y_{1\ldots N}]]$. When written in this form, the probability is referred to as a *Gibbs distribution.* The terms $\psi_c[y_{1\ldots N}]$ are functions that may return any real number and can be thought of as representing a cost for every combination of labels $y_{1\ldots N}$. As the cost increases, the probability decreases.

11.2.1 Connecting neighbours with undirected models

In this chapter, we will consider a subset of undirected models called *Markov Random Fields* (MRFs). The key idea is that the potential functions $\phi[\bullet]$ each address only a small subset of the variables, and these subsets are carefully chosen to induce simple conditional independence relations. Before providing a formal definition, let's consider a simple concrete example.

Consider five variables where we define the probability $Pr(y_{1\ldots 5})$ over the associated discrete states as a normalized product of pairwise terms:

$$Pr(y_{1\ldots 5}) = \frac{1}{Z}\phi_{12}(y_1,y_2)\phi_{23}(y_2,y_3)\phi_{34}(y_3,y_4)\phi_{45}(y_4,y_5), \tag{11.6}$$

where $\phi_{mn}(y_m,y_n)$ are potential functions that take the two states y_m and y_n and return a positive number.

This distribution has a very important property: each variable is conditionally independent of all of the others given the variables with neighbouring indices. For example, the dependence of the state y_3 can be written as

$$Pr(y_3|y_1,y_2,y_4,y_5) = Pr(y_3|y_2,y_4). \tag{11.7}$$

This relationship is not obvious from the original definition (equation 11.6) but can be easily proved using the conditional probability relation,

$$\begin{aligned} Pr(y_3|y_1, y_2, y_4, y_5) &= \frac{Pr(y_1, y_2, y_3, y_4, y_5)}{Pr(y_1, y_2, y_4, y_5)} \\ &= \frac{Pr(y_1, y_2, y_3, y_4, y_5)}{\sum_{y_3} Pr(y_1, y_2, y_3, y_4, y_5)} \end{aligned} \tag{11.8}$$

where we the probability $Pr(y_1, y_2, y_4, y_5)$ in the denominator of the first line is found by marginalizing the full joint density $Pr(y_1, y_2, y_3, y_4, y_5)$ with respect to y_3. Now, substituting in the original expression for the joint density we get

$$\begin{aligned} \frac{Pr(y_1, y_2, y_3, y_4, y_5)}{\sum_{y_3} Pr(y_1, y_2, y_3, y_4, y_5)} &= \frac{\frac{1}{Z}\phi_{12}(y_1, y_2)\phi_{23}(y_2, y_3)\phi_{34}(y_3, y_4)\phi_{45}(y_4, y_5)}{\sum_{y_3} \frac{1}{Z}\phi_{12}(y_1, y_2)\phi_{23}(y_2, y_3)\phi_{34}(y_3, y_4)\phi_{45}(y_4, y_5)} \\ &= \frac{\phi_{23}(y_2, y_3)\phi_{34}(y_3, y_4)}{\sum_{y_3} \phi_{23}(y_2, y_3)\phi_{34}(y_3, y_4)}, \end{aligned} \tag{11.9}$$

where the final expression depends only on the neighbors y_2 and y_4 as required. Hence the links in the graphical model (figure 11.2) tell us about conditional probability relations: a variable is conditionally independent from all the other variables given its neighbors. By making different choices for the functions $\phi()$ we can model distributions with different conditional independence structure.

Let's consider the situation where the world state y_n at each pixel is binary and so takes values 0 or 1. The positive function ϕ_{mn} will now return four possible values depending on which of the four configurations $\{00, 01, 10, 11\}$ of y_n and y_m is present. For simplicity, we'll assume that the functions $\phi_{12}, \phi_{23}, \phi_{34}$ and ϕ_{45} are identical and that

$$\begin{aligned} \phi_{mn}(0,0) &= 1.0 & \phi_{mn}(0,1) &= 0.1 \\ \phi_{mn}(1,0) &= 0.1 & \phi_{mn}(1,1) &= 1.0. \end{aligned} \tag{11.10}$$

Since there are only five binary states, we can calculate the constant Z explicitly by computing the product of the functions for each of the 32 possible combinations and taking the sum as in equation 11.4. The resulting probabilities for each of the 32 possible states are shown below:

$y_{1...5}$	$Pr(y_{1...5})$	$y_{1...5}$	$Pr(y_{1...5})$	$y_{1...5}$	$Pr(y_{1...5})$	$y_{1...5}$	$Pr(y_{1...5})$
00000	0.34151	01000	0.00342	10000	0.03415	11000	0.03415
00001	0.03415	01001	0.00034	10001	0.00342	11001	0.00342
00010	0.00342	01010	0.00003	10010	0.00034	11010	0.00034
00011	0.03415	01011	0.00034	10011	0.00342	11011	0.00342
00100	0.00342	01100	0.00342	10100	0.00034	11100	0.03415
00101	0.00034	01101	0.00034	10101	0.00003	11101	0.00342
00110	0.00342	01110	0.00342	10110	0.00034	11110	0.03415
00111	0.03415	01111	0.03415	10111	0.00342	11111	0.34151

The costs defined in equation 11.10 encourage smoothness in the labels: the functions ϕ_{mn} return higher values when the neighbours take the same state and

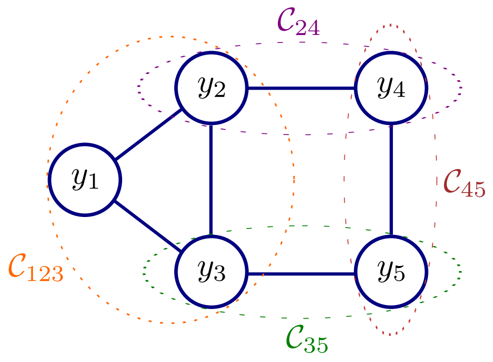

Figure 11.3 Maximal cliques. Cliques are subsets of sites, where every variable is connected to every other. In a maximal clique it is not possible to add any further variables and still be a clique. The maximal cliques are shown for a system of five variables $y_{1\ldots5}$. For example, variables y_1, y_2, y_3 are all connected to each other, but there are no other variables which are connected to all three, and so they form the maximal clique $\mathcal{C}_{123}$.

lower values when they differ and this is reflected in the resulting probabilities. This model captures similar properties to the directed models in the previous chapter.

It should be noted that for more realistically sized problems it will not be able to compute the normalizing constant Z by brute force as above. For example, with 10,000 pixels each taking 20 values, the normalizing constant is the sum of 20^{10000} terms. In general we have to cope with only knowing the probabilities up to an unknown scaling factor.

This worked example should have provided some insight into the properties of MRFs, so now we proceed to a more formal description.

11.2.2 Markov Random Fields

A Markov Random Field is determined by

- a set of sites $\mathcal{S} = \{1 \ldots N\}$. These will correspond to the N pixel locations,
- a set of random variables $y = \{y_1 \ldots y_N\}$ associated with each of the sites,
- a set of neighbors $\mathcal{N}_{1\ldots N}$ at each of the N sites.

To be a Markov random field, the model must obey the Markov property,

$$Pr(y_n|y_{\mathcal{S}\backslash n}) = Pr(y_n|y_{\mathcal{N}_n}) \qquad \forall \quad n \in \mathcal{S}, \tag{11.11}$$

that states that y_n is conditionally independent of all of the other variables given the values of the variables in the neighborhood $\mathcal{N}_n$.

It can be shown (via the impressively named *Hammersley-Clifford theorem*) that any function that obeys the Markov property (equation 11.11) can be written as a Gibbs distribution so that

$$Pr(\mathbf{y}) = \frac{1}{Z} \exp\left[-\sum_{c\in\mathcal{C}} \psi_c(\mathbf{y})\right]. \tag{11.12}$$

The terms ψ_c are called *clique potentials* and are effectively costs. As the clique potential increases, the probability decreases. There is one clique potential for every *maximal clique* in the graph. A clique is a subset of the sites, where every pair of

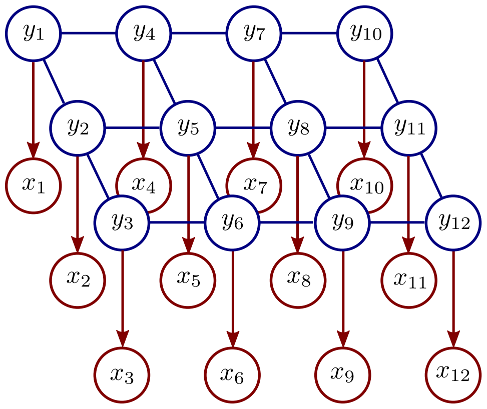

Figure 11.4 Undirected model for images. The observed data $\mathbf{x}_n$ at pixel n is conditionally dependent on the associated world state y_n (red directed edges). Each world state y_n has undirected edges to its four-connected neighbors (blue undirected edges). Together the world states are connected in a Markov random field with cliques that consist of neighbouring pairs of variables. For example variable y_5 contributes to cliques $\mathcal{C}_{25}, \mathcal{C}_{45}, \mathcal{C}_{65}, \mathcal{C}_{85}$

variables is connected to every other (figure 11.3). For a clique to be maximal it must not be possible to add any further variables and remain a clique.

As before term Z is an unknown constant factor which normalizes this to make it a proper probability distribution and is known as the *partition function.* We can only write it as as the sum

$$Z = \sum_{y} \exp\left[-\sum_{c\in\mathcal{C}} \psi_c(\mathbf{y})\right]. \tag{11.13}$$

11.2.3 Image denoising with discrete pairwise MRFs

Recall that in the binary image denoising task (section 11.1) the likelihood of observing pixel x_n given the original underlying image y_n is given by

$$\begin{aligned} Pr(x_n|y_n=0) &= \text{Bern}_{x_n}[\rho] \\ Pr(x_n|y_n=1) &= \text{Bern}_{x_n}[1-\rho] \end{aligned} \tag{11.14}$$

We now define the prior probability over the world state to be a Markov random field, with cliques $\mathcal{C}$. In this chapter we will restrict the form of the MRF do have *pairwise* connections only (figure 11.4). In other words, each clique consists of two neighboring variables, and each 4-connection contributes one clique so that

$$Pr(y_{1\ldots N}) = \frac{1}{Z}\exp\left[-\sum_{(m,n)\in\mathcal{C}} \psi(y_m, y_n, \boldsymbol{\theta})\right]. \tag{11.15}$$

where we have assumed that the clique potentials $\psi()$ are the same for every (y_m, y_n). The parameters$\boldsymbol{\theta}$ define the costs $\psi()$ for each combination of neighboring pairwise values,

$$\psi(y_m = j, y_n = k, \boldsymbol{\theta}) = \theta_{jk}, \tag{11.16}$$

so when the first variable y_m in the clique takes label j and the second variable y_n takes label k we pay a price of θ_{jk}. In general we will choose these values so that there is a small cost when neighboring labels are the same (so θ_{00} and θ_{11} are small) and a larger one when the neighboring labels differ (so θ_{01} and θ_{10} are large). This has the effect of encouraging solutions that are mostly smooth.

The remaining part of the chapter concerns inference and learning in this model. In section 11.3 we describe how to perform inference (estimate the original noiseless image) by drawing samples from the posterior. In section 11.4 we describe methods to find the maximum a posteriori estimate of the original image. Finally, in section 11.5 we describe how to learn the parameters of the MRF prior from training examples.

11.3 Inference by Drawing Samples

We now turn our attention to the problem of infering the world states $y_{1\ldots N}$ from the observed data $x_{1\ldots N}$ (i.e. inference). Unfortunately, the MRF is not conjugate (section 2.9) to the likelihood model and there is no way to express the full posterior distribution over the states in closed form.

In this section we consider drawing samples from the posterior distribution over these states. This is not the most practical mode of inference but it is always possible even in situations where the maximum a posteriori (MAP) solution cannot be computed. In section 11.4 we will return to MAP estimation of the world states.

According to Bayes' rule, the posterior probability over the states is proportional to the product of the likelihood terms and the prior

$$Pr(y_{1\ldots N}|\mathbf{x}_{1\ldots N}) \propto \prod_{n=1}^{N} Pr(x_n|y_n).Pr(y_{1\ldots N}) \tag{11.17}$$

For the equivalent directed model (section 10.1), each variable y_n had a specified set of parents $y_{pa[n]}$ and a well defined conditional probability relations $Pr(y_n|y_{pa[n]})$. This allows us to draw samples using the ancestral technique (section 10.1.4). Unfortunately, there is no sense that each variable has parents in the Markov random field and so ancestral sampling is not possible.

An alternative approach to generating samples from complex high-dimensional probability distributions is to use a *Markov chain Monte Carlo* (MCMC) method. The principle is to generate a series (chain) of samples from the distribution, so that each sample depends directly on the previous one (hence "Markov"). However, the generation of the sample is not completely deterministic (hence, "Monte Carlo").

11.3.1 Gibbs Sampling

The MCMC method that we will use is called *Gibbs sampling.* This is a method that can be used to draw samples from any high dimensional probability distribution.

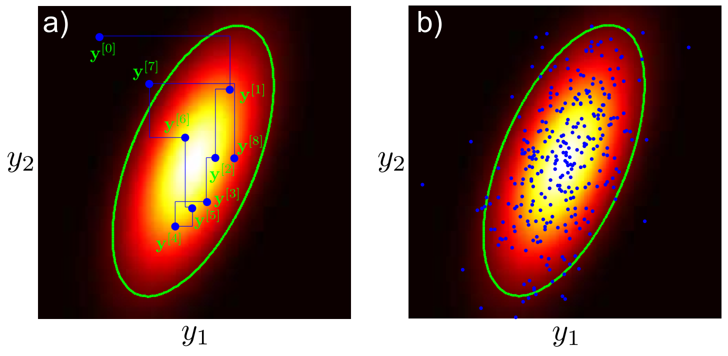

Figure 11.5 Gibbs sampling. We generate a chain of samples, by cycling through each dimension in turn and drawing a sample from the conditional distribution of that dimension given that the others are fixed. a) For this 2d multivariate normal distribution we start at a random position $y^{[0]}$. We alternately draw samples from the conditional distribution of the first dimension keeping the second fixed (horizontal changes) and the second dimension keeping the first fixed (vertical changes). For the multivariate normal, these conditional distributions are themselves normal (section 4.5). Each time we cycle through all of the dimensions, we create a new sample $y^{[n]}$. b) Many samples generated using this method.

We'll start by discussing sampling from the MRF prior $Pr(y_{1...N})$ before showing how to draw samples from the posterior.

The method is as follows. First, we randomly choose the initial state $\mathbf{y}^{[1]}$ using any method. We generate the next sample in the chain $\mathbf{y}^{[2]}$ by updating the state at each of the dimensions in turn. In the MRF, each dimension corresponds to one pixel site. To update the nth dimension y_n, we fix the other $N-1$ dimensions and draw from the conditional distribution $Pr(y_n|y_{1...N\backslash n})$. After we have modified every dimension in this way, we have the second sample in the chain. This idea is illustrated in figure 11.5 for the multivariate normal distribution.

When this procedure is repeated a very large number of times, so that the initial conditions are forgotten, a sample from this sequence can be considered as a draw from the $Pr(y_{1...N})$. Although this is not immediately obvious (a proof is beyond the scope of this book) this procedure does clearly have some sensible properties: since we are sampling from the conditional probability distribution at each pixel, we are more likely to change the current label to a configuration with overall higher probability. The system will gradually tend towards more probable configurations. However, the stochastic update rule provides the possibility of (infrequently) visiting less probable regions of the space.

For the MRF case, we calculate the conditional distribution for the nth dimension (pixel site) keeping the remaining sites constant using the conditional probability relation,

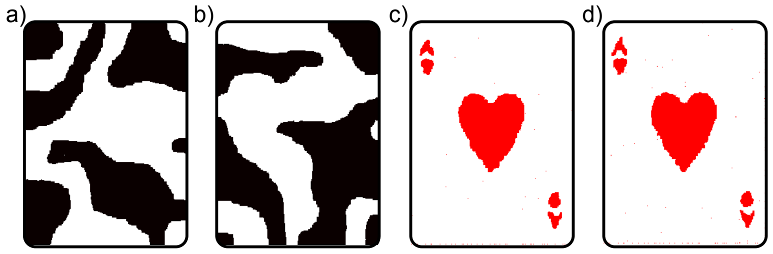

Figure 11.6 Gibbs sampling results. a-b) Samples from prior. This MRF encourages solutions that are mostly smooth c-d) Samples from posterior in denoising task (compare to figure 11.1b).

$$Pr(y_n = k|y_{1\ldots N\backslash n}) = \frac{Pr(y_n = k, y_{1\ldots N\backslash n})}{Pr(y_{1\ldots N\backslash n})} = \frac{Pr(y_n = k, y_{1\ldots N\backslash n})}{\sum_{k=1}^{K} Pr(y_n = k, y_{1\ldots N\backslash n})}. \tag{11.18}$$

We then randomly choose the next value of y_n according to this categorical distribution. In practice this can be computed very efficiently, because of the Markov property of the distribution (equation 11.11): in practice almost all of the potentials in the numerator and denominator do not involve variable y_n and cancel out. Examples of samples from this type of prior are illustrated in figure 11.6a-b.

To sample from the posterior distribution, we use a similar formulation

$$\begin{aligned}
Pr(y_n = k|y_{1\ldots N\backslash n}, x_{1\ldots N}) &= \frac{Pr(y_n = k, y_{1\ldots N\backslash n}, x_{1\ldots N})}{\sum_{k=1}^{K} Pr(y_{n=k}, y_{1\ldots N\backslash n}, x_{1\ldots N})} \qquad (11.19)\\
&= \frac{\prod_{i\in 1\ldots N\backslash N} Pr(x_i|y_i)Pr(x_n|y_n = k)Pr(y_n = k, y_{1\ldots N\backslash n})}{\sum_{k=1}^{K}\prod_{i\in 1\ldots N\backslash N} Pr(x_i|y_i)Pr(x_n|y_n = k)Pr(y_n = k, y_{1\ldots N\backslash n})}\\
&= \frac{Pr(x_n|y_n = k)Pr(y_n = k, y_{1\ldots N\backslash n})}{\sum_{k=1}^{K} Pr(x_n|y_n = k)Pr(y_n = k, y_{1\ldots N\backslash n})},
\end{aligned}$$

where we have used Bayes' rule in both the numerator and denominator in the second line.

Note that the Gibbs sampling procedure only requires us to know the probability up to a constant scaling factor: we can hence draw samples from high dimensional distributions where it is intractable to compute the normalizing constant Z.

Gibbs sampling could be used to search the state space for a solution with a high posterior probability. Two possible solutions (draws from the posterior distribution) for the binary denoising task are shown in figure 11.6c-d. Alternately, we could approximate the marginal posterior distributions by taking many samples and looking at the frequency of each label at each pixel location. In either case, this method is rather slow and impractical, although it is applicable for all MRFs.

In the following section we investigate MAP inference. As we shall see, efficient algorithms for this task only exist for restricted subsets of MRF problems.

11.4 MAP Inference

In MAP inference, we aim to find the set of world states $\hat{y}_{1\ldots N}$ that maximizes the posterior probability $Pr(y_{1\ldots N}|\mathbf{x}_{1\ldots N})$ so that

$$\begin{aligned}\hat{y}_{1\ldots N} &= \arg\max_{y_{1\ldots N}} Pr(y_{1\ldots N}|\mathbf{x}_{1\ldots N})\\ &= \arg\max_{y_{1\ldots N}} \prod_{n=1}^{N} Pr(x_n|y_n)Pr(y_{1\ldots N})\\ &= \arg\max_{y_{1\ldots N}} \sum_{n=1}^{N} \log[Pr(x_n|y_n)] + \log[Pr(y_{1\ldots N})]. \end{aligned} \tag{11.20}$$

where we have applied Bayes' rule, assumed that the observations at each site were independent, and transformed to the log domain. For the case where the prior is a binary MRF with pairwise connections we compute

$$\begin{aligned}\hat{y}_{1\ldots N} &= \arg\max_{y_{1\ldots N}} \sum_{n=1}^{N} \log[Pr(x_n|y_n)] - \sum_{(m,n)\in\mathcal{C}} \psi(y_m, y_n, \boldsymbol{\theta})\\ &= \arg\min_{y_{1\ldots N}} \sum_{n=1}^{N} U_n(y_n) + \sum_{(m,n)\in\mathcal{C}} P_{mn}(y_m, y_n) \end{aligned} \tag{11.21}$$

where $U_n(y_n)$ denotes the *unary* term at pixel n. This is a cost for observing the data at pixel n given the state was y_n and is due to the negative log likelihood term. Similarly, $P_{mn}(y_m, y_n)$ denotes the *pairwise* term. This is a cost for placing labels y_m and y_n at neighboring locations m and n and is due to the clique costs $\psi(y_m, y_n, \boldsymbol{\theta})$ from the MRF prior (which are assumed to be identical at each location). Note that we have omitted the term $-\log[Z]$ from the MRF definition as it is constant with respect to the states $y_{1\ldots N}$ and hence does not effect the optimal solution.

The cost function in equation 11.21 can be optimized using a set of techniques known collectively as *graph cuts*. We will consider three cases:

- binary MRFs (i.e. $y_i \in \{0, 1\}$) where the costs for different combinations of adjacent labels are "submodular". Exact MAP inference is tractable here.
- multi-label MRFs (i.e. $y_i \in \{1, 2\ldots, K\}$) where the costs are "submodular". Once more, exact MAP inference is possible.
- multi-label MRFs where the costs are more general. Exact MAP inference is intractable, but good approximate solutions can be found in some cases.

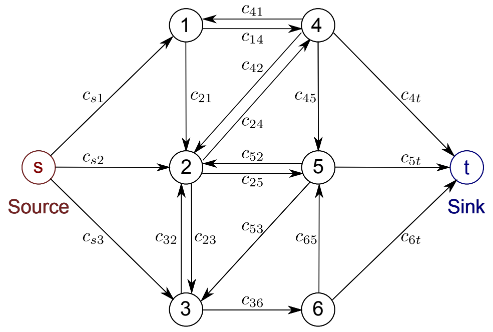

Figure 11.7 Max flow problem: we are given a network of vertices connected by directed edges, each of which has a non-negative capacity c_{mn}. There are two special vertices s and t termed the source and sink respectively. In the max flow problem we seek to push as much 'flow' from source to sink while respecting the capacities of the edges.

To solve these MAP inference tasks we will translate them into the form of *maximum flow* problems. Maximum flow (or max-flow) problems are well-studied and exact polynomial time algorithms exist. In the following section we describe the max-flow problem and its solution. In sections 11.4.2-11.4.4 we describe how to translate MAP inference for binary and multi-label Markov Random Fields into max-flow problems.

11.4.1 Max-flow / Min-cut

Consider a graph $\mathbf{G} = \{\mathcal{V}, \mathcal{E}\}$ with vertices $\mathcal{V}$ and directed edges $\mathcal{E}$ connecting them (figure 11.7). Each edge has a non-negative *capacity* so that the edge between vertices m and n has capacity c_{mn}. Two of the vertices are treated as special and are termed the *source* and the *sink*.

Consider transferring some quantity ('flow') through the network from the source to the sink. The goal of the max-flow algorithm is to compute the maximum amount of flow that can be transferred across the network without exceeding any of the edge capacities.

When the maximum possible flow is being transferred, every path from source to sink must include a saturated edge (one where the capacity is reached) - if not then we could push more flow down this path and so by definition this is not the maximum flow solution.

It follows that an alternate way to think about the problem is to consider the edges that saturate. We define a 'cut' on the graph to be a minimal set of edges, that when removed prevent flow from the source from the sink. A cut hence partitions the nodes into two groups: nodes that can be reached by some path from the source, but cannot reach the sink, and nodes that cannot be reached from the source, but can reach the sink via some route. For short, we will refer a cut as 'separating' the source from the sink. Every cut is given an associated cost which is the sum of the capacities of the excised edges.

Since the saturated edges in the max flow solution separate the source from the sink they form a cut. In fact, this particular choice of cut has the minimum possible cost and is referred to as the *min-cut* solution. Hence, the maximum flow and minimum cut problems can be considered interchangeably.

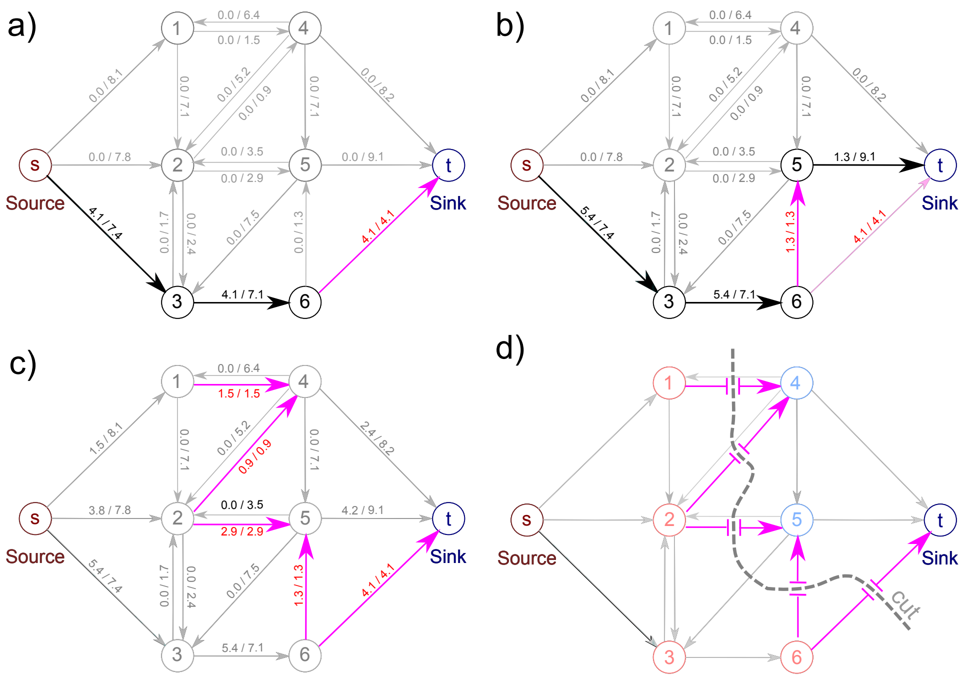

Figure 11.8 Augmenting paths algorithm for max-flow. Numbers attached edges correspond to current flow / capacity. a) We choose any route from source to sink with spare capacity and push as much flow as possible along this route. The edge with the smallest capacity (here edge 6-t) saturates. b) We then choose another route where there is still spare capacity and push as much flow as possible. Now edge 6-5 saturates. c) We repeat this until there is no route from source to sink that does not contain a saturated edge (highlighted edges are saturated). The total flow pushed is the maximum flow. d) In the min-cut problem, we seek a set of edges that separate the source from the sink and have minimal total capacity. The min-cut (dashed gray line) consists of the saturated edges in the max-flow problem.

Augmenting paths algorithm for maximum flow

There are many algorithms to compute the maximum flow, and to describe them properly is beyond the scope of this volume. However, for completeness, we present a sketch of the *augmenting paths* algorithm (figure 11.8).

Consider choosing any path from the source to sink and pushing the maximum possible amount of flow along it. This flow will be limited by the edge on that path that has the smallest capacity which will duly saturate. We remove this amount of flow from the capacities of all of the edges along the path and causing the saturated edge to have a new capacity of zero. We repeat this procedure, finding a second path from source to sink, pushing as much flow as possible along it and updating the capacities. We continue this process until there is no path from source to sink without at least one saturated edge. The total flow that we have transferred is the

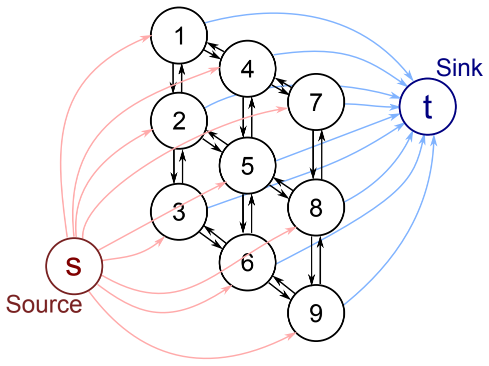

Figure 11.9 Graph structure for finding MAP solution for MRF with binary labels and pairwise connections in a 3×3 image. There is one vertex per pixel and neighbors in the pixel grid are connected by reciprocal pairs of directed edges. Each pixel vertex receives a connection from the source and passes one to the sink. To separate source from sink, the cut must include one of these two edges for each vertex. The choice of which edge is cut will determine which of two labels is assigned to the pixel.

maximum flow, and the saturated edges form the minimum cut. There are some extra complications: for example, if there is already some flow along edge $i-j$, it may be that there is a remaining path from source to sink that includes the edge $j-i$. In this situation we reduce the flow in $i-j$ before adding flow to $j-i$. The reader should consult a specialized text on graph-based algorithms for more details.

If we choose the path with the greatest remaining capacity at each step, the algorithm is guaranteed to converge and has complexity $O(|\mathcal{E}|^2|\mathcal{V}|)$ where $|\mathcal{E}|$ is the number of edges and $|\mathcal{V}|$ the number of vertices in the graph. From now on we will assume that the max-flow/min-cut problem can be solved and concentrate on how to convert MAP estimation problems with MRFs into this form.

11.4.2 MAP inference: binary variables

Recall that to find the MAP solution we must find

$$\hat{y}_{1\ldots N} = \arg\min_{y_{1\ldots N}} \sum_{n=1}^{N} U_n(y_n) + \sum_{(m,n)\in\mathcal{C}} P_{mn}(y_m, y_n) \qquad (11.22)$$

where $U_n(y_n)$ denotes the *unary* term and $P_{mn}(y_m, y_n)$ denotes the *pairwise* term.

For pedagogical reasons, we will first consider cases where the unary terms are positive and the pairwise terms have the following zero-diagonal form

$$\begin{aligned} P_{mn}(0,0) &= 0 \qquad & P_{mn}(1,0) &= \theta_{10} \\ P_{mn}(0,1) &= \theta_{01} \qquad & P_{mn}(1,1) &= 0, \end{aligned}$$

where $\theta_{01}, \theta_{10} > 0$. We discuss using more general values of $\boldsymbol{\theta}$ in section 11.4.2.

The key idea will be to set up a directed graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ and attach weights to the edges, so that the minimum cut on this graph corresponds to the maximum a posteriori solution. In particular, we construct a graph with one vertex per pixel, and a pair of directed edges between each adjacent vertices in the pixel grid. In

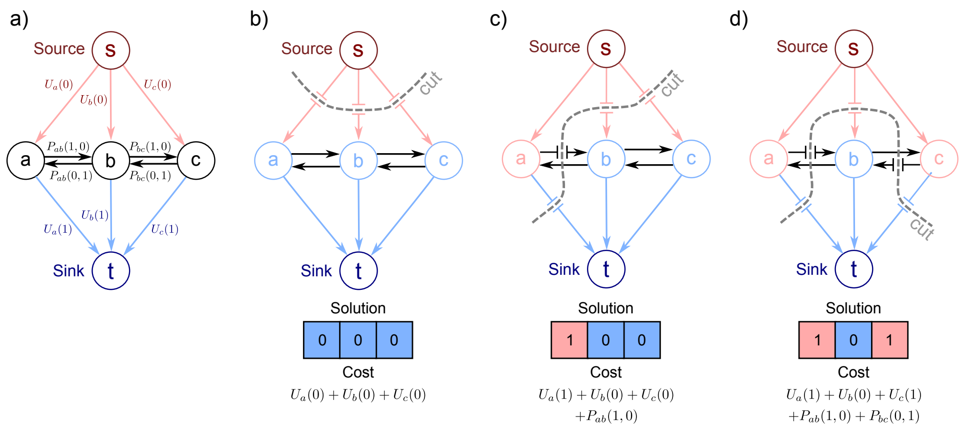

Figure 11.10 Graph construction for binary MRF with diagonal pairwise terms using simple 1d example. a) After the cut, vertices attached to the source are given label 1 and vertices attached to the sink are given label 0. When we attach unary and pairwise terms to the edges as shown here, the correct cost is paid for each solution. b) For example, the solution $(a=0, b=0, c=0)$ requires us to cut edges $s-a, s-b, s-c$ and pay the cost $U_a(0)+U_b(0)+U_c(0)$. c) For the solution $(a=1, b=0, c=0)$ we must cut edges $a-t, s-b, s-c$ and $a-b$ (to prevent flow through the route $s-a-b-t$). This incurs a total cost of $U_a(1)+U_b(0)+U_c(0)+P_{ab}(1,0)$. d) Similarly, in this example with $(a=1, b=0, c=1)$, we pay the appropriate cost $U_a(1)+U_b(0)+U_c(1)+P_{ab}(1,0)+P_{bc}(0,1)$. This construction is readily extended to higher-dimensional graphs such as that of figure 11.2.

addition, there is a directed edge from the source to every vertex and a directed edge from every vertex to the sink (figure 11.9).

Now consider a cut on the graph. In any cut we must either remove the edge that connects the source to a pixel vertex or the edge that connects the pixel vertex to the sink or both. If we do not do this, then there will still be a path from source to sink and it is not a valid cut. For the minimum cut, we will never cut both - this is unnecessary and will inevitably incur a greater cost than cutting one or the other. We'll label pixels where the edge to the source was cut as $y_n = 0$ and pixels where the edge to the sink was cut as having label $y_n = 1$. So each plausible minimum cut is associated with a pixel labeling.

Our goal is now to assign capacities to the edges the edges so the cost of each cut matches the cost of the associated labelling as prescribed in equation 11.22. For simplicity, we illustrate these with a 1d image with 3 pixels (figure 11.10a), but we stress that all the ideas are also valid for 2d images.

We attach the unary costs $U_n(0)$ and $U_n(1)$ to the edges from the pixel to the source and sink respectively. If we cut the edge between a pixel to the source (and hence assign $y_n = 0$) we pay the cost $U_n(0)$. Conversely, if we cut the edge to the sink (and hence assign $y_n = 1$) we pay the cost $U_n(1)$.

We attach the pairwise costs $P_{mn}(1,0)$ and $P_{mn}(0,1)$ to the two edges between

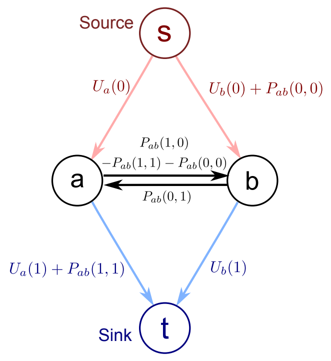

Figure 11.11 Graph structure for general (i.e. non-diagonal) pairwise costs. Consider the solution $(a = 0, b = 0)$. We must break the edges $s-a$ and$s-b$ giving a total cost of $U_a(0) + U_b(0) + P_{ab}(0,0)$. For the solution $(a = 1, b = 0)$ we must break the edges $a-t, a-b$ and $s-b$ giving a total cost of $U_a(1)+U_b(0)+P_{ab}(1,0)$. Similarly, the cuts corresponding to the solutions $(a = 0, b = 1)$ and $(a = 1, b = 1)$ on this graph have pairwise costs $P_{ab}(0,1)$ and $P_{ab}(1,1)$ respectively.

adjacent pixels. Now if one pixel is attached to the source and the other to the sink, we pay either $P_{mn}(0,1) = \theta_{01}$ or $P_{mn}(1,0) = \theta_{10}$ as appropriate to separate source from sink as in figures 11.10c-d.

Any cut on the graph in which each pixel is either separated from the source or the sink now has the appropriate cost from equation 11.22. It follows that the minimum cut on this graph will have the minimum cost and the associated labeling $y_{1...N}$ will correspond to the maximum a posteriori solution.

General Pairwise Costs

Now let's consider how to use the more general pairwise costs,

$$\begin{aligned} P_{mn}(0,0) &= \theta_{00} \qquad & P_{mn}(1,0) &= \theta_{10} \\ P_{mn}(0,1) &= \theta_{01} \qquad & P_{mn}(1,1) &= \theta_{11}. \end{aligned} \tag{11.23}$$

To illustrate this, we use an even simpler graph with only two pixels (figure 11.11). Notice that we have added the pairwise cost $P_{ab}(0,0)$ to the edge $s-b$ - we will have to pay this cost appropriately in configuration where $y_a = 0$ and $y_b = 0$. Unfortunately, we would also pay it in the case where $y_a = 1$ and $y_b = 0$. Hence, we subtract the same cost from the edge $b–a$ which must also be cut in this solution. By a similar logic we add $P_{ab}(1,1)$ to the edge $a-t$ and subtract it from edge $b-a$. In this way we associate the correct costs with each labeling.

Reparameterization

The above discussion assumed that the edge costs are all non-negative and can be valid capacities in the max-flow problem. Unfortunately, this is not generally necessarily the case. Even if the original unary and pairwise terms were positive, the edge $a-b$ in figure 11.11 with cost $P_{ab}(1,0) - P_{ab}(1,1) - P_{ab}(0,0)$ could be negative. The solution to this problem is *reparameterization.*

The goal of reparameterization is to modify the costs associated with the edges in the graph in such a way that the MAP solution is not changed. In particular, we will change the edge capacities so that every possible solution has a constant cost added to it. This does not change which solution has the minimum cost, and so the MAP labelling will be unchanged.

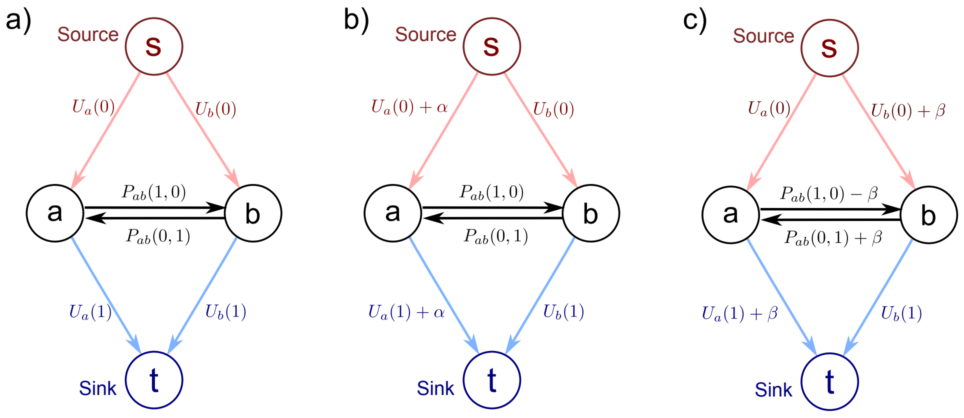

Figure 11.12 Reparameterization. a) Original graph construction. b) Reparameterization 1. Adding a constant cost α to the connections from a pixel vertex to both the source and sink results in a problem with the same MAP solution. Since we must cut either, but not both of these edges, every solution increases in cost by α, and the minimum cost solution remains the same. c) Reparameterization 2. Manipulating the edge capacities in this way results in a constant β being added to every solution and so the choice of minimum cost solution is unaffected. The easiest way to convince yourself of this is to write out the costs for all four solutions.

We consider two reparameterizations (figure 11.12). First, consider adding a constant cost α to the edge from a given pixel to the source and the edge from the same pixel to the sink. Since any solution cuts exactly one of these edges, the overall cost of every solution increases by α. We can use this to ensure that none of the edges connecting the pixels to the source and sink have negative costs: we simply add a sufficiently large positive value α to make them non-negative.

A more subtle type of reparameterization is illustrated in figure 11.12c. By changing the costs in this way, we increase the value of each possible solution by β. Applying this reparameterization to the general construction in figure 11.11, we must ensure that the capacities on edges between pixel nodes are positive so that

$$\begin{aligned} \theta_{10}-\theta_{11}-\theta_{00}-\beta &\geq 0 && (11.24)\\ \theta_{01}+\beta &\geq 0. && (11.25) \end{aligned}$$

Adding these equations together, we can eliminate β to get a single inequality

$$\theta_{01}+\theta_{10}-\theta_{11}-\theta_{00}\geq 0. \quad (11.26)$$

If this condition holds, the problem is termed *submodular* and the graph can be reparameterized to have only non-negative costs. It can then can be solved in polynomial time using the max-flow algorithm. If the condition does not hold, then this approach cannot be used and in general the problem is NP hard. Fortunately,

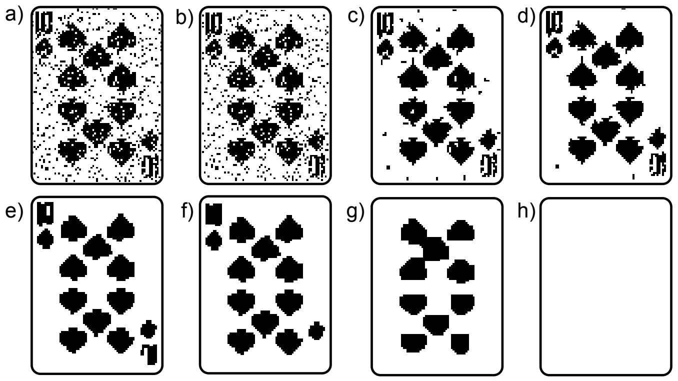

Figure 11.13 Denoising results. a) Observed noisy image. b-h) Maximum a posteriori solution as we increase zero-diagonal pairwise costs. When the pairwise costs are low, the unary terms dominate and the MAP solution is the same as the observed image. As the pairwise costs increase the image gets more and more smooth until eventually it becomes a uniform field, ignoring the unary contributions entirely.

the former case is much more common for vision problems: we generally favour solutions where neighbouring labels are the same and hence the costs θ_{01}, θ_{10} for labels differing are naturally greater than the costs θ_{00}, θ_{11} for the labels agreeing.

Figure 11.13 shows the MAP solutions to the binary denoising problem with an MRF prior as we increase the strength of the cost for having adjacent labels that differ. Here we have assumed that the costs θ_{01}, θ_{10} when neighboring nodes differ are the same and there is no cost when neighboring labels are the same ($\theta_{00}, \theta_{11} = 0$) so we are in the 'zero-diagonal' regimen. When the MRF costs are small, the solution is dominated by the unary terms and the MAP solution looks like the noisy image. As the costs increase the solution ceases to tolerate isolated regions and most of the noise is removed. When the costs become larger details such as the centre of the '0' in '10' are lost and eventually nearby regions are connected together. With very high pairwise costs, the MAP solution is a uniform field of labels: the overall cost is dominated by the pairwise terms from the MRF and the unary terms have no effect.

11.4.3 MAP inference: multi-valued variables

We now investigate MAP inference using MRF priors with pairwise connections when the world state y_n at each pixel can take multiple labels $\{1, 2, \ldots K\}$. To solve the multilabel problem, we change the graph construction (figure 11.14a). With K labels and N pixels, we introduce $(K + 1) \times N$ vertices into the graph.

For each pixel, the $K+1$ associated vertices are stacked and the top and bottom of the stack are connected to the source and sink by edges with infinite capacity.

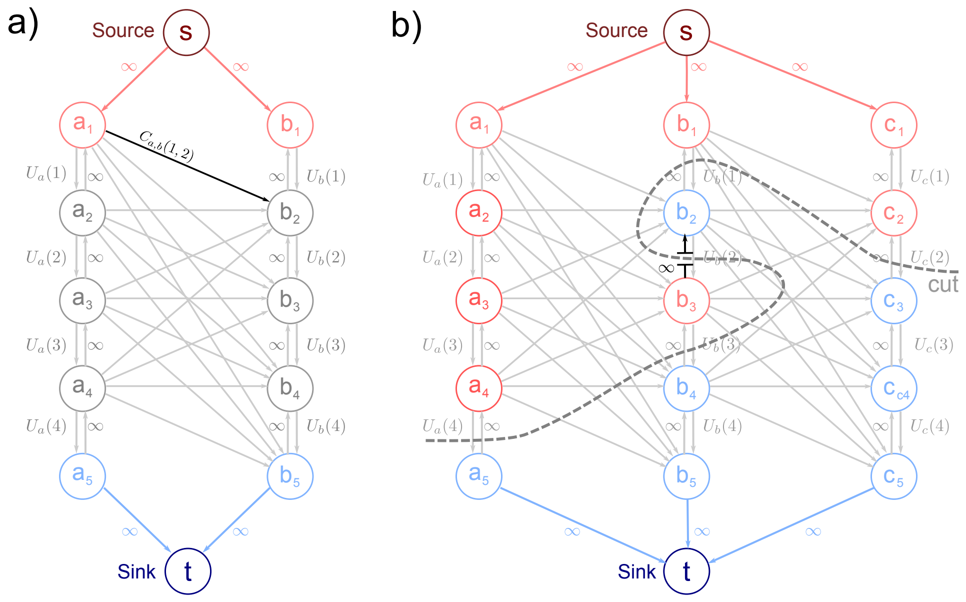

Figure 11.14 a) Graph setup for multi-label case for two pixels (a, b) and four labels $(1, 2, 3, 4)$. There is a chain of 5 vertices associated with each pixel. The four vertical edges between these vertices are assigned the unary costs for the four labels. The minimum cut must break this chain to separate source from sink, and the label is assigned according to where the chain is broken. Vertical constraint edges of infinite capacity run between the four vertices in the opposite direction. There are also diagonal edges between the i'th vertex of pixel a and the j'th vertex of pixel b with assigned costs $C_{ab}(i, j)$ (see text). b) The vertical constraint edges prevent solutions like this, where the chain of vertices associated with a pixel is cut in more than one place. For this to happen a constraint link must be cut and hence this solution has an infinite cost.

Between the $K + 1$ vertices in the stack are K edges forming a path from source to sink. These edges are associated with the K unary costs $U_n(1) \ldots U_n(K)$. To separate the source from the sink, we must cut at least one of the K edges in this chain. We will interpret a cut at the kth edge in this chain as indicating that the pixel takes label k and this incurs the appropriate cost of $U_n(k)$.

To ensure that only a single edge from the chain is part of the minimum cut (and hence that each possible cut corresponds to one valid labelling), we add *constraint edges*. These are edges of infinite capacity that connect the vertices backwards along each chain. Any cut that crosses the chain more than once must cut one of these edges and will never be the minimum cut solution (figure 11.14b).

In figure 11.14a, there are also inter-pixel edges from the vertices associated with pixel a to the vertices associated with pixel b. These are assigned costs $C_{ab}(i, j)$ where i indexes the vertex associated with pixel a and j indexes the vertex associated with pixel b. We choose the edges costs to be

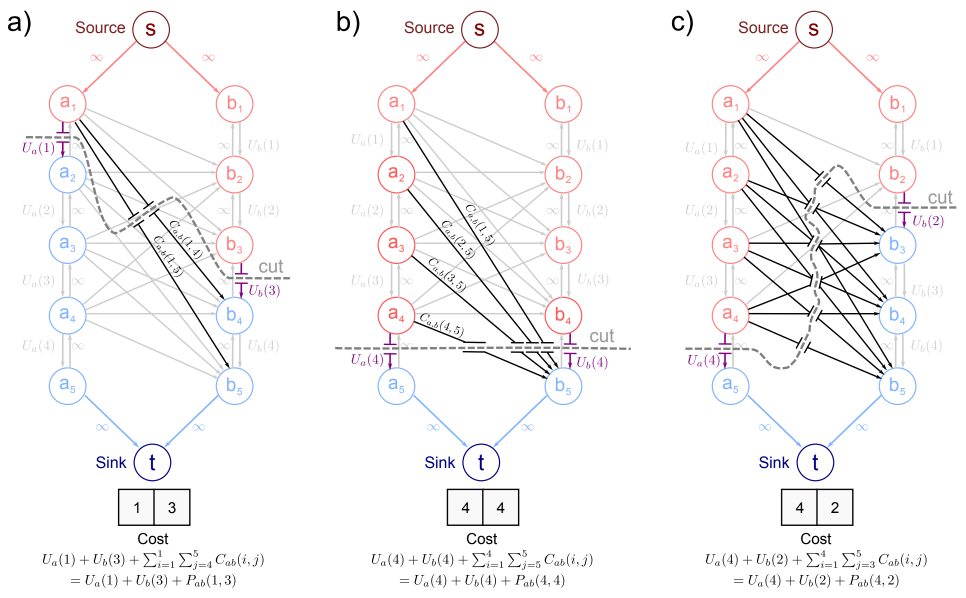
a)
Source
s
Sink
t
cut
1 3
Cost
$U_a(1)+U_b(3)+\sum_{i=1}^{1}\sum_{j=4}^{5}C_{ab}(i,j)$
$=U_a(1)+U_b(3)+P_{ab}(1,3)$
b)
Source
s
Sink
t
cut
4 4
Cost
$U_a(4)+U_b(4)+\sum_{i=1}^{4}\sum_{j=5}^{5}C_{ab}(i,j)$
$=U_a(4)+U_b(4)+P_{ab}(4,4)$
c)
Source
s
Sink
t
cut
4 2
Cost
$U_a(4)+U_b(2)+\sum_{i=1}^{4}\sum_{j=3}^{5}C_{ab}(i,j)$
$=U_a(4)+U_b(2)+P_{ab}(4,2)$

Figure 11.15 Example cuts for multi-label case. To separate the source and sink, we must cut all of the links that pass from above the chosen label for pixel a to below the chosen label for pixel b. a) Pixel a is set to label 1 and pixel b is set to label 3 meaning we must cut the links from vertex a_1 to nodes b_4 and b_5. b) Pixel a takes label 4 and pixel b takes label 4. c) Pixel a takes label 4 and pixel b takes label 2.

$$C_{ab}(i,j) = P_{ab}(i,j-1) + P_{ab}(i-1,j) - P_{ab}(i,j) - P_{ab}(i-1,j-1), \qquad (11.27)$$

where we define any superfluous pairwise costs associated with the non-existent labels 0 or $K+1$ to be zero, so that

$$\begin{aligned} P_{ab}(i,0) &= 0 \qquad P_{ab}(i,K+1) = 0 \qquad \forall\ i \in \{0\ldots K+1\} \\ P_{ab}(0,j) &= 0 \qquad P_{ab}(K+1,j) = 0 \qquad \forall\ j \in \{0\ldots K+1\} \qquad (11.28) \end{aligned}$$

When label I is assigned to pixel a and label J to pixel b we must cut all of the links from vertices $a_1 \ldots a_I$ to the vertices $b_{J+1} \ldots b_{K+1}$ to separate the source from the sink (figure 11.15). So, the total cost due to the inter-pixel edges for assigning label I to pixel a and label J to pixel b is

$$\begin{aligned} \sum_{i=1}^{I}\sum_{j=J+1}^{K+1} C_{ab}(i,j) &= \sum_{i=1}^{I}\sum_{j=J+1}^{K+1} P_{ab}(i,j-1)+P_{ab}(i-1,j)-P_{ab}(i,j)-P_{ab}(i-1,j-1) \\ &= P_{ab}(I,J)+P_{ab}(0,K+1)-P_{ab}(I,K+1)-P_{ab}(0,K+1) \\ &= P_{ab}(I,J), \qquad (11.29) \end{aligned}$$

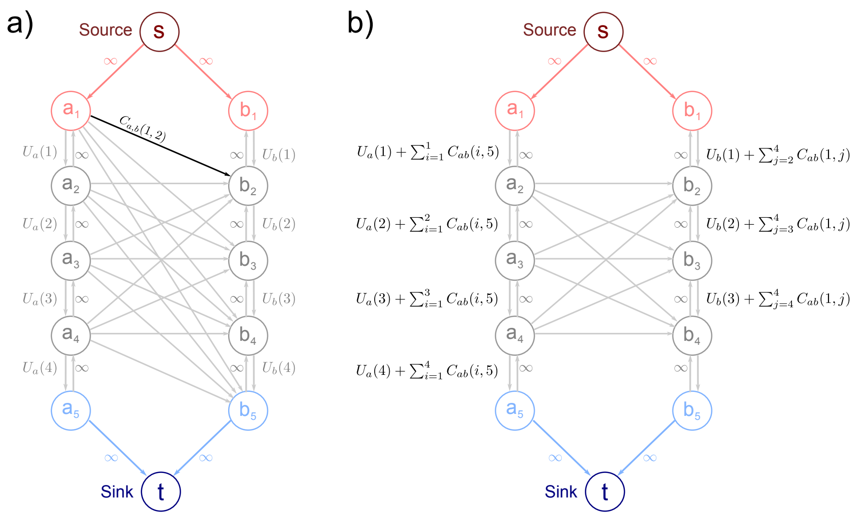
a)
Source
s
a1
b1
a2
b2
a3
b3
a4
b4
a5
b5
Sink
t
$C_{ab}(1,2)$
$U_a(1)$
$U_a(2)$
$U_a(3)$
$U_a(4)$
$U_b(1)$
$U_b(2)$
$U_b(3)$
$U_b(4)$
∞
b)
$U_a(1)+\sum_{i=1}^{1} C_{ab}(i,5)$
$U_a(2)+\sum_{i=1}^{2} C_{ab}(i,5)$
$U_a(3)+\sum_{i=1}^{3} C_{ab}(i,5)$
$U_a(4)+\sum_{i=1}^{4} C_{ab}(i,5)$
$U_b(1)+\sum_{j=2}^{4} C_{ab}(1,j)$
$U_b(2)+\sum_{j=3}^{4} C_{ab}(1,j)$
$U_b(3)+\sum_{j=4}^{4} C_{ab}(1,j)$

Figure 11.16 Reparameterization for multi-label graph cuts. The original construction (a) is equivalent to construction (b). The label at pixel b determines which edges that leave node a_1 are cut. Hence, we can remove these edges and add the extra costs to the vertical links associated with pixel b. Similarly, the costs of the edges passing into node b_5 can be added to the vertical edges associated with pixel a. If any of the resulting vertical edges associated with a pixel are non-negative, we can add a constant α to each: since exactly one is broken, the total cost increases by α but the MAP solution remains the same.

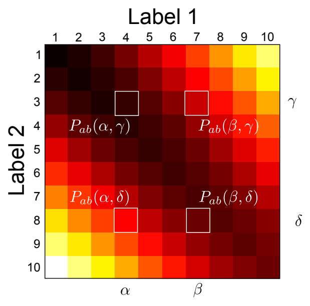

Figure 11.17 Submodularity constraint for multi-label case (colour indicates cost pairwise costs $P_{ab}(m,n)$. For all edges in the graph to be positive, we require that the pairwise terms obey the constraint $P_{ab}(\beta,\gamma)+P_{ab}(\alpha,\delta)-P_{ab}(\beta,\delta)-P_{ab}(\alpha,\gamma)\geq 0$ for all $\alpha,\beta,\gamma,\delta$. If this condition holds, the problem can be solved in polynomial time, otherwise the problem is NP-complete.

Adding the unary terms, the total cost is $U_a(I)+U_b(J)+P_{ab}(I,J)$ as required.

Once more, we have implicitly made the assumption that the costs associated with edges are non-negative. If the vertical (intra-pixel) edges terms have negative costs, it is possible to re-parameterize the graph by adding a constant α to all of the unary terms. Since the final cost includes exactly one unary term per pixel, every possible solution increases by α and the MAP solution is unaffected.

The diagonal inter-pixel edges are more problematic. It is possible to remove the edges that leave node a_1 and the edges that arrive at b_K by adding terms to the

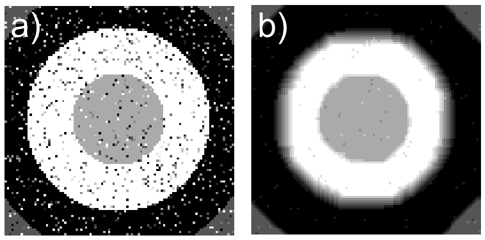

Figure 11.18 Denoising results with convex (quadratic) pairwise costs. a) Noisy observed image. b) Denoised image has artefacts where there are large intensity changes in the original image. Convex costs imply that there is a lower cost for a number of small changes rather than a single large one.

intra-pixel edges associated with the unary terms (figure 11.16). These intra-pixel edges can then be reparameterized as described above if necessary. Unfortunately, we can neither remove nor re-parameterize the remaining inter-pixel edges so we require that

$$C_{ab}(i,j) = P_{ab}(i,j-1) + P_{ab}(i-1,j) - P_{ab}(i,j) - P_{ab}(i-1,j-1) \geq 0 \quad (11.30)$$

By mathematical induction we get the more general result (figure 11.17),

$$P_{ab}(\beta,\gamma) + P_{ab}(\alpha,\delta) - P_{ab}(\beta,\delta) - P_{ab}(\alpha,\gamma) \geq 0, \quad (11.31)$$

where $\alpha, \beta, \gamma, \delta$ are any four values of the state y such that $\beta > \alpha$ and $\gamma > \delta$. This is the multi-label generalization of the submodularity condition (equation 11.26). An important class of pairwise costs that are submodular are those which are convex in the absolute difference $|y_i - y_j|$ between the labels at adjacent pixels (figure 11.19a). Here, smoothness is encouraged as the penalty becomes increasingly stringent as the jumps between labels increase.

Unfortunately, convex potentials are not always appropriate. For example, in the denoising task we might expect the image to be piecewise smooth: there are smooth regions (corresponding to objects) followed by abrupt jumps (corresponding to the boundaries between objects). A convex potential function cannot describe this situation, because it penalizes large jumps much more than smaller ones. The result is that the MAP solution smooths over the sharp edges changing the label by several smaller amounts rather than one large jump (figure 11.18).

To solve this problem, we need to work with interactions that are non-convex in the absolute label difference, such as the truncated quadratic function or the Pott's model (figures 11.19b-c). These favor small changes in the label, and penalize large changes equally or nearly equally. This reflects the fact that the exact size of an abrupt jump in label is relatively unimportant. Unfortunately, these pairwise costs do not satisfy the submodularity constraint (equation 11.31). Here, the MAP solution cannot in general be found exactly with the method described above, and the problem is NP-hard. Fortunately, there are good approximate methods for optimizing such problems, one of which is the alpha-expansion algorithm.

11.4.4 Inference: alpha expansion

The *alpha expansion* algorithm works by breaking the solution down into a series of binary problems, each of which can be solved exactly. At each iteration we choose

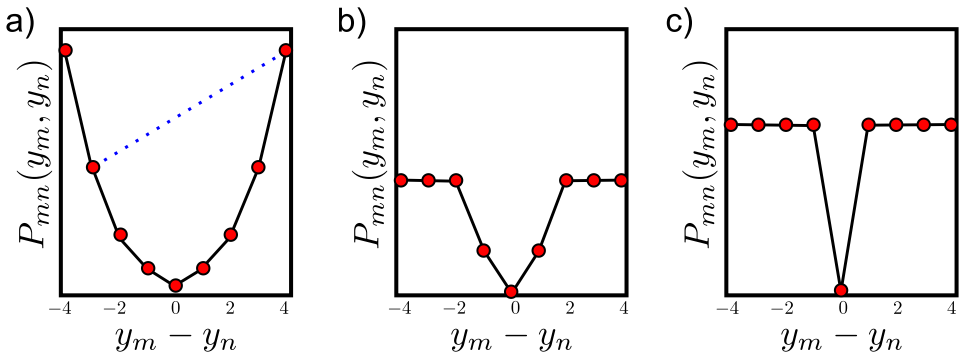

Figure 11.19 Convex vs. non-convex potentials. The method for MAP inference for multi-valued variables depends on whether the costs are a convex or non-convex function of the difference in labels. a) Quadratic function (convex), $P_{mn}(y_m, y_n) = \kappa(y_m - y_n)^2$. For convex functions, it is possible to draw a chord between any two points on the function without intersecting the function anywhere between (e.g. dashed blue line). b) Truncated quadratic function (non-convex), $P_{mn}(y_m, y_n) = \min(\kappa_1, \kappa_2(y_m - y_n)^2)$. c) Potts model (non-convex), $P_{mn}(y_m, y_n) = \kappa(1 - \delta(y_m - y_n))$.

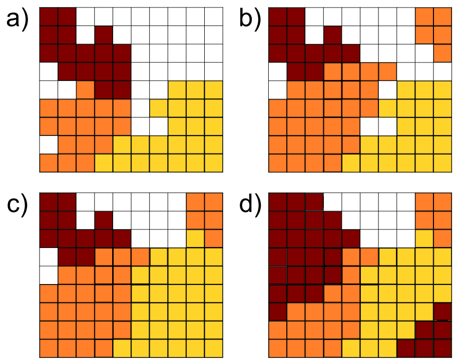

Figure 11.20 The alpha expansion algorithm breaks the problem down into a series of binary sub-problems. At each step, we choose a label α and we *expand*: for each pixel we either leave the label as it is or replace it with α. This sub-problem is solved in such a way that it is guaranteed to decrease the multilabel cost function. a) Initial labelling. b) Orange label is expanded: each label stays the same or becomes orange. c) Yellow label is expanded. d) Red label is expanded.

one label value α, and for each pixel we consider either retaining the current label, or switching it to α. The name alpha-expansion derives from the fact that the real estate occupied by label α in the solution expands at each iteration (figure 11.20). The process is iterated until no choice of α causes any change. Each expansion move is guaranteed to lower the overall objective function although the final result is not guaranteed to be the global minimum.

For the alpha expansion algorithm to work, we require that the edge costs form a metric. In other words, we require that

$$P(\alpha,\beta) = 0 \quad \Leftrightarrow \alpha = \beta \tag{11.32}$$
$$P(\alpha,\beta) = P(\beta,\alpha) > 0 \tag{11.33}$$
$$P(\alpha,\beta) \leq P(\alpha,\gamma) + P(\gamma,\beta). \tag{11.34}$$

These assumptions are reasonable for many applications in vision, and allow us to model non-convex priors.

In the alpha-expansion graph construction (figure 11.21), there is one vertex associated with each pixel. Each of these vertices is connected to the source (representing keeping the original label or $\overline{\alpha}$) and the sink (representing the label α). To separate source from sink, we must cut one of these two edges at each pixel. The choice of edge will determine whether we keep the original label or set it to α. Accordingly, we associate the unary costs for each edge being set to α or its original label with the two links from each pixel. If the pixel already has label α then we set the cost of being set to $\overline{\alpha}$ to ∞.

The remaining structure of the graph is dynamic: it changes at each iteration depending on the choice of α and the original labels. There are four possible relationships between adjacent pixels:

- Pixel i has label α and the pixel j has label α. Here, the final configuration is inevitably $\alpha - \alpha$, and so the pairwise cost is zero and there is no need to add further edges connecting nodes i and j in the graph. Pixels a and b in figure 11.21 have this relationship.

- The first pixel has label α but the second pixel has a different label β. Here the final solution may be $\alpha - \alpha$ with zero cost or $\alpha - \beta$ with cost $P_{ij}(\alpha,\beta)$. Here we add a single edge connecting pixel j to pixel i with cost $P_{ij}(\alpha,\beta)$. Pixels b and c in figure 11.21 have this relationship.

- Both pixels i and j take the same label β. Here the final solution may be β–β with zero cost, α–β with cost $P_{ij}(\alpha,\beta)$ or β–α with cost $P_{ij}(\beta,\alpha)$. We add two edges between the pixel representing the two non-zero costs. Pixels c and d in figure 11.21 have this relationship.

- Pixel i takes label β and pixel j takes a second label γ. Here the final solution may be α–α with zero cost, β–γ with cost $P_{ij}(\beta,\gamma)$, β–α with cost $P_{ij}(\beta,\alpha)$ and $\alpha - \gamma$ with cost $P_{ij}(\alpha,\gamma)$. Here we add a new vertex k between vertices i and j and add these three non zero costs to edges $k - t$, $i - k$ and $k - j$ respectively. Pixels e and f in figure 11.21 have this relationship.

Note that this construction critically relies on the triangle inequality (equation 11.34). If this does not hold and $P_{ij}(\beta,\gamma) > P_{ij}(\beta,\alpha) + P_{ij}(\alpha,\gamma)$ then when both pixel nodes are attached to the sink (as for pixels d and e in figure 11.21b), it will be cheaper to cut both the links $d - k$ and $e - k$ than to cut $k - \alpha$ and the wrong cost will be paid. In practice, it is sometimes possible to ignore this constraint by truncating the offending cost $P_{ij}(\beta,\gamma)$ and running the algorithm as normal. After the cut is done, the true objective function can be computed for the new label map and the answer accepted if the cost has decreased.

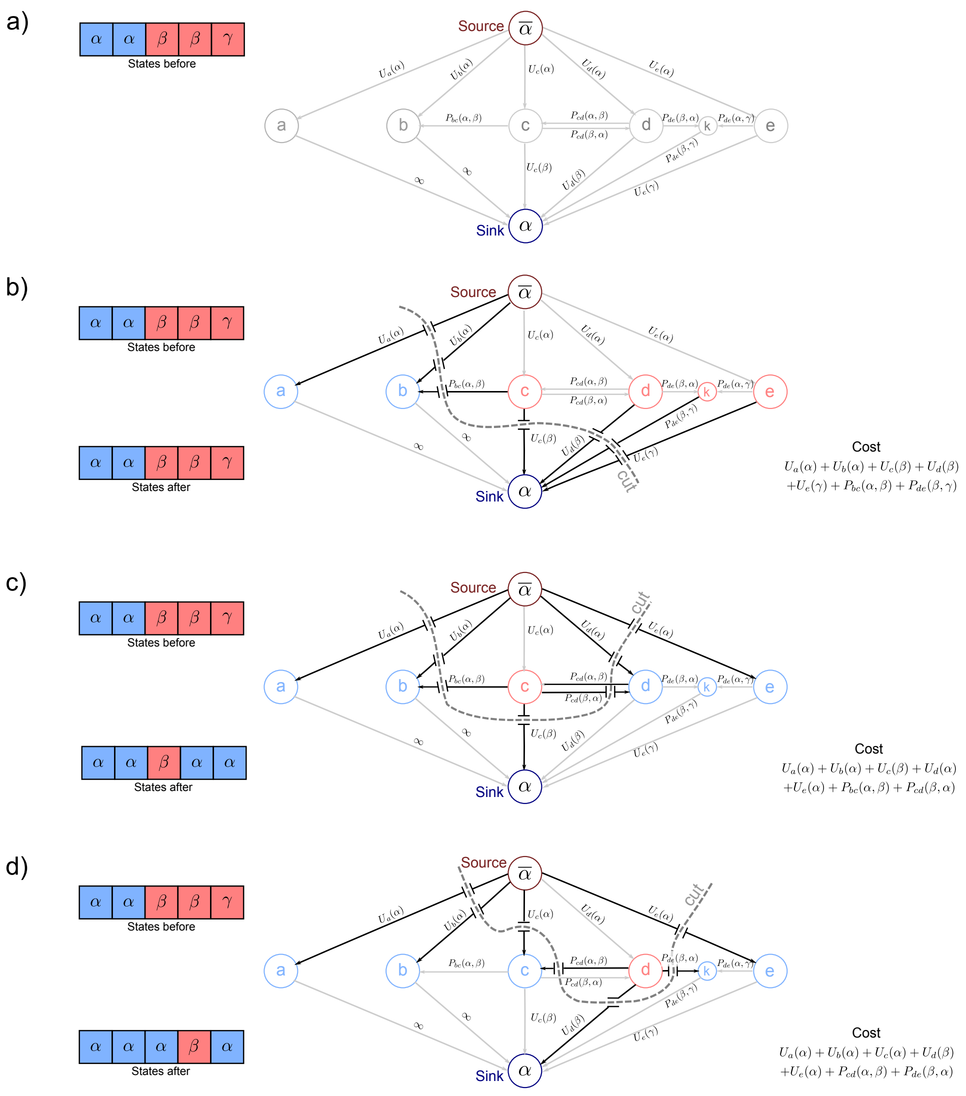
a)
b)
c)
d)
States before
States after
Source
Sink
cut
$\overline{\alpha}$
α
Cost
$U_a(\alpha)+U_b(\alpha)+U_c(\beta)+U_d(\beta)+U_e(\gamma)+P_{bc}(\alpha,\beta)+P_{de}(\beta,\gamma)$
Cost
$U_a(\alpha)+U_b(\alpha)+U_c(\beta)+U_d(\alpha)+U_e(\alpha)+P_{bc}(\alpha,\beta)+P_{cd}(\beta,\alpha)$
Cost
$U_a(\alpha)+U_b(\alpha)+U_c(\alpha)+U_d(\beta)+U_e(\alpha)+P_{cd}(\alpha,\beta)+P_{de}(\beta,\alpha)$

Figure 11.21 a) Alpha expansion graph setup. Each pixel node (a,b,c,d,e) is connected to the source and the sink by edges which have costs $U_{\bullet}(\overline{\alpha})$ and $U_{\bullet}(\alpha)$ respectively. In the minimum cut exactly one of these links will be cut. The nodes and vertices describing the relationship between neighbouring pixels depends on their current labels, which may be $\alpha-\alpha$ as for pixels a and b, $\alpha-\beta$ as for pixels b and c, $\beta-\beta$ as for pixels c and d or $\beta-\gamma$ as for pixels d and e. For the last case an auxiliary node k must be added to the graph. b-d) Example cuts on this graph illustrate that the appropriate unary and pairwise costs are always paid.

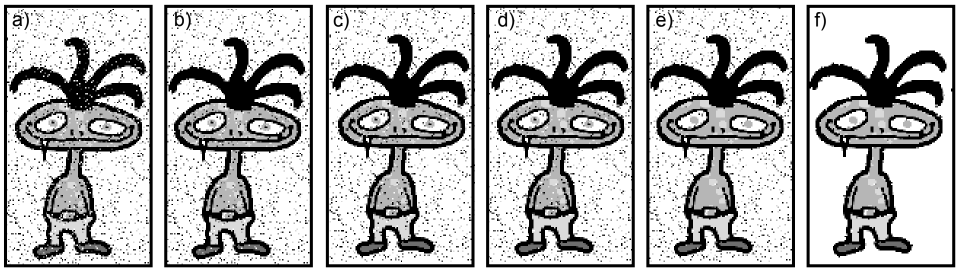

Figure 11.22 Alpha expansion algorithm for denoising task. a) Observed noisy image. b) Label 1 (black) is expanded, removing noise from the hair. c-f) Subsequent iterations in which the labels corresponding to the boots, trousers, skin and background are expanded respectively.

It should be emphasized that although each step optimally updates the objective function with respect to expanding α, this algorithm is not guaranteed to converge to the overall global minimum. However, it can be proven that the result is within a factor of two of the minimum and often it behaves much better.

Figure 11.27 shows an example of multi-label denoising using the alpha expansion algorithm. On each iteration one of the labels is chosen and is expanded and the appropriate region is denoised. Sometimes the label is not supported at all by the unary costs and nothing happens. The algorithm terminates when no choice of α causes any further change.

11.5 Learning MRF models

It is possible to set the parameters $\boldsymbol{\Theta}$ that specify the costs for adjacent labels in the Markov random field prior by hand to encourage smoothness. However, a more principled method is to learn the appropriate values from training data. Given I training label fields $\mathbf{y}_{1\ldots I}$ and we aim to fit parameters $\boldsymbol{\Theta}$. Assuming that the training examples are independent, the maximum likelihood solution is

$$\begin{aligned}\hat{\boldsymbol{\Theta}} &= \arg\max_{\boldsymbol{\Theta}} \frac{1}{Z(\boldsymbol{\Theta})^I} \exp\left[-\sum_{i=1}^{I}\sum_{c\in\mathcal{C}} \psi_c(\mathbf{y}_i, \boldsymbol{\Theta})\right] \\ &= \arg\max_{\boldsymbol{\Theta}} -I\log[Z(\boldsymbol{\Theta})] - \sum_{i=1}^{I}\sum_{c\in\mathcal{C}} \psi_c(\mathbf{y}_i, \boldsymbol{\Theta}), \end{aligned} \tag{11.35}$$

where as usual we have taken the log to simplify the expression.

To maximize this expression we calculate the derivative of the log likelihood L with respect to the parameters $\boldsymbol{\Theta}$,

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{\Theta}} &= -I\frac{\partial \log[Z(\boldsymbol{\Theta})]}{\partial \boldsymbol{\Theta}} - \sum_{i=1}^{I}\sum_{c\in\mathcal{C}}\frac{\partial \psi_c(\mathbf{y}_i,\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}} \qquad (11.36)\\ &= -I\frac{\partial \log\left[\sum_{\mathbf{y}_i}\exp\left[-\sum_{c\in\mathcal{C}}\psi_c(\mathbf{y}_i,\boldsymbol{\Theta})\right]\right]}{\partial \boldsymbol{\Theta}} - \sum_{i=1}^{I}\sum_{c\in\mathcal{C}}\frac{\partial \psi_c(\mathbf{y}_i,\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}}\end{aligned}$$

The second term is readily computable, but the first term involves an intractable sum over all possible states: we cannot compute the derivative with respect to the parameters and so learning is difficult. We can neither find an algebraic solution, nor can we hope to use an optimization technique to steadily move uphill. The best that we can do is to find an approximation to the gradient.

11.5.1 Contrastive divergence

One possible solution to this problem is the *contrastive divergence* algorithm. This is a method for approximating the gradient of the log likelihood with respect to parameters $\boldsymbol{\theta}$ for functions with the general form,

$$Pr(\mathbf{y}) = \frac{1}{Z(\boldsymbol{\theta})}f(\mathbf{y},\boldsymbol{\theta}), \qquad (11.37)$$

where $Z(\boldsymbol{\theta}) = \sum_{\mathbf{y}} f(\mathbf{y},\boldsymbol{\theta})$ is the normalizing constant and the derivative of the log likelihood is

$$\frac{\partial \log[Pr(\mathbf{y})]}{\partial \boldsymbol{\theta}} = -\frac{\partial \log[Z(\boldsymbol{\theta})]}{\partial \boldsymbol{\theta}} + \frac{\partial \log[f(\mathbf{y},\boldsymbol{\theta})]}{\partial \boldsymbol{\theta}}. \qquad (11.38)$$

The main idea behind contrastive divergence follows from some algebraic manipulation of the first term:

$$\begin{aligned}\frac{\partial \log[Z(\boldsymbol{\theta})]}{\partial \boldsymbol{\theta}} &= \frac{1}{Z(\boldsymbol{\theta})}\frac{\partial Z(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\\ &= \frac{1}{Z(\boldsymbol{\theta})}\frac{\partial \sum_{\mathbf{y}} f(\mathbf{y},\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\\ &= \frac{1}{Z(\boldsymbol{\theta})}\sum_{\mathbf{y}}\frac{\partial f(\mathbf{y},\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\\ &= \frac{1}{Z(\boldsymbol{\theta})}\sum_{\mathbf{y}} f(\mathbf{y},\boldsymbol{\theta})\frac{\partial \log[f(\mathbf{y},\boldsymbol{\theta})]}{\partial \boldsymbol{\theta}}\\ &= \sum_{\mathbf{y}} Pr(\mathbf{y})\frac{\partial \log[f(\mathbf{y},\boldsymbol{\theta})]}{\partial \boldsymbol{\theta}}. \qquad (11.39)\end{aligned}$$

The last term is the expectation of the derivative of $\log[f()]$. We cannot compute this exactly, but we can approximate it by drawing J independent samples $\mathbf{y}^*$ from the distribution to yield

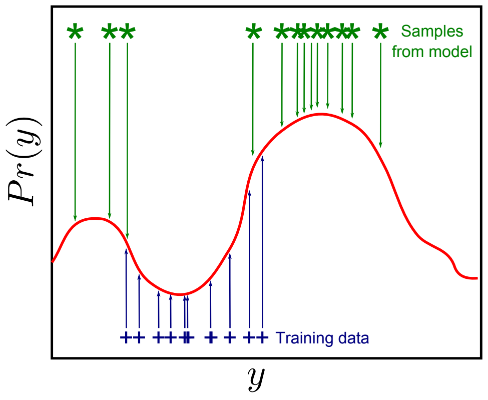

Figure 11.23 The contrastive divergence algorithm changes the parameters so that the un-normalized distribution increases at the observed data points (blue crosses) but decreases at sampled data points from the model. These two components counterbalance one another and ensure that the likelihood increases. When the model fits the data these two forces will cancel out and the parameters will remain constant.

$$\frac{\partial \log[Z(\boldsymbol{\theta})]}{\partial \boldsymbol{\theta}} = \sum_{\mathbf{y}} Pr(\mathbf{y}) \frac{\partial \log[f(\mathbf{y}, \boldsymbol{\theta})]}{\partial \boldsymbol{\theta}} \approx \frac{1}{J} \sum_{j=1}^{J} \frac{\log[f(\mathbf{y}_j^*, \boldsymbol{\theta})]}{\partial \boldsymbol{\theta}}. \tag{11.40}$$

With I training examples $\mathbf{y}_{1...I}$, the gradient of the log likelihood, L is hence

$$\frac{\partial L}{\partial \boldsymbol{\theta}} \approx -\frac{I}{J} \sum_{j=1}^{J} \frac{\partial \log[f(\mathbf{y}_j^*, \boldsymbol{\theta})]}{\partial \boldsymbol{\theta}} + \sum_{i=1}^{I} \frac{\partial \log[f(\mathbf{y}_i, \boldsymbol{\theta})]}{\partial \boldsymbol{\theta}} \tag{11.41}$$

A visual explanation of this expression is presented in figure 11.23. The gradient points in a direction that (i) increases the logarithm of the unnormalized function at the data points $\mathbf{y}_i$ but (ii) decreases the same quantity in places where the model believes the density is high (i.e. the samples $\mathbf{y}_j^*$). When the model fits the data these two forces will cancel out that the parameters will stop changing.

This algorithm requires us to draw samples y^* from the model at each iteration of the optimization procedure in order to compute the gradient. Unfortunately, the only way to draw samples from MRF models is to use costly Markov Chain Monte Carlo methods such as Gibbs sampling (section 11.3.1) and this is impractically time consuming. In practice it has been found that even approximate samples will do: one method is to re-start I samples at the datapoints at each iteration and do just a few MCMC steps. Surprisingly, this works well even with a single MCMC step. Another approach is to start with the previous samples at each iteration and perform a few MCMC steps: here the samples are free to wander without restarting. This technique is known as *persistent* contrastive divergence.

11.6 Conditional Random Fields

The Markov random fields presented above form the prior in a generative model of the image data. For the equivalent discriminative model the posterior probability of the world state (label field) $\mathbf{y}$ given the observed data $\mathbf{x}$ is written as

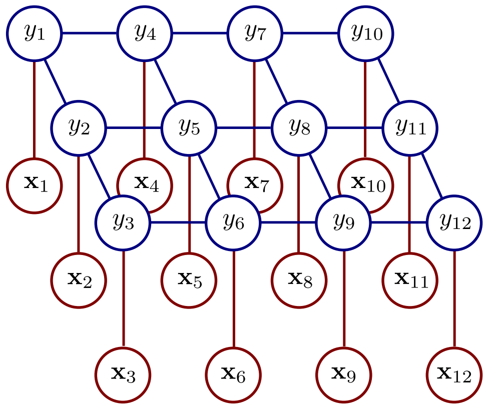

Figure 11.24 Graphical model for conditional random field (compare to figure 11.4). The posterior probability of the labels $\mathbf{y}$ is a Markov random field for fixed data $\mathbf{x}$. In this model, the cliques relate (i) neighbouring labels and (ii) each label to its associated measurement. Since this model only includes unary and pairwise interactions between the labels it can be optimized using graph cut techniques.

$$Pr(\mathbf{y}|\mathbf{x}) = \frac{1}{Z}\exp\left[-\sum_c \psi_C(\mathbf{y}) - \sum_d \zeta(\mathbf{y},\mathbf{x})\right] \tag{11.42}$$

where the functions $\psi(\bullet)$ enforce structure in the label field and the functions $\zeta(\bullet)$ enforce agreement between the data and the label field. If we condition on the data (and hence treat it as a fixed quantity) then this has the same mathematical form as a Markov random field. Hence this model is called a *conditional random field*.

If the functions $\psi(\bullet)$ are used to encourage smoothness between neighbouring labels and the functions $\zeta(\bullet)$ each relate the compatibility of one label y_n to its associated measurement x_n then the negative log posterior probability will again be the sum of unary and pairwise terms. The best labels $\hat{\mathbf{y}}$ can hence be found by minimizing the cost function

$$\hat{\mathbf{y}} = \arg\min_{y_{1\ldots N}} \sum_{n=1}^{N} U_n(y_n) + \sum_{(m,n)\in\mathcal{C}} P_{mn}(y_m, y_n) \tag{11.43}$$

and the graphical model will be as in figure 11.24. This cost function can be minimized using the graph cuts techniques described throughout this chapter.

11.7 Applications

Markov random fields are used in many areas of computer vision including stereo vision, motion estimation, background subtraction, segmentation, image editing and building 3d models. Here we review a few key applications.

11.7.1 Segmentation

Pairwise Markov Random Field priors have been used in many segmentation applications where the label field is binary and reflects the presence or absence of an

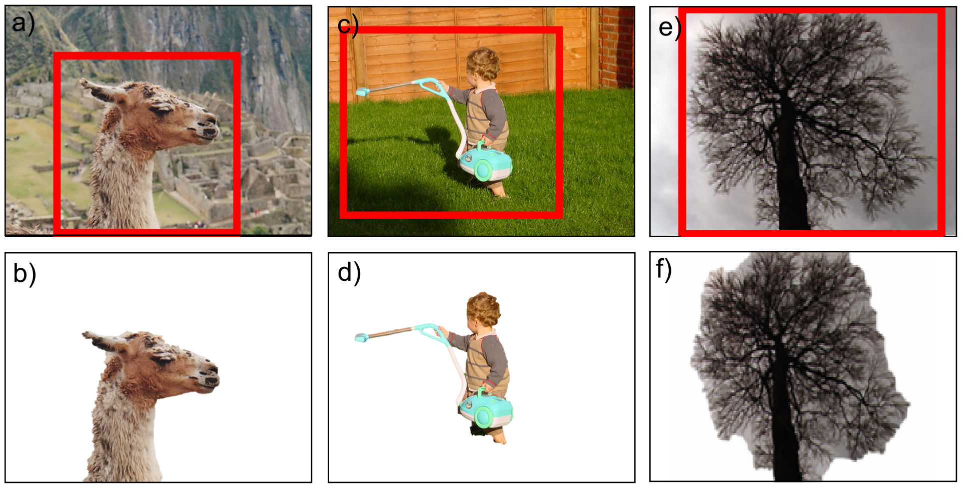

Figure 11.25 Grab Cut. a) The user draws a bounding box around the object of interest. b) The algorithm segments the foreground from the background. c-d) Another example. e-f) Failure mode. This algorithm does not segment 'wiry' objects well as the pairwise costs for tracing around all the boundaries are prohibitive.

object. For example, the skin segmentation and background subtraction methods from chapter 6 can be improved using this technique.

Binary MRFs have also been used for interactive segmentation. In the *Grab-cut* algorithm, the user draws a rectangular box around the object of interest. The system then builds two mixtures of Gaussians models to describe the likelihood of observing the RGB data at each pixel given that the object is part of the foreground or the background. On the first iteration, the background model is learnt from regions of the image close to the edges of the rectangular box and the foreground model is learnt from the centre. The MAP segmentation is then computed. The likelihood models are relearned from the new segmentation and the system iterates until it reaches convergence.

To improve the performance of this algorithm it is possible to modify the MRF so that the pairwise cost for changing from foreground to background label is less where there is an edge in the image. This is referred to as using "geodesic distance". From a pure probabilistic viewpoint, this is somewhat dubious as the MRF prior should embody what we know about the task before seeing the data and hence cannot depend on the image. However, this is largely a philosophical objection, and the method works well in practice for a wide variety of objects.

A notable failure mode is in segmenting 'wiry' object such as trees. Here the model is not prepared to pay the extensive pairwise costs to cut exactly around the many edges of the object and so the segmentation is poor.

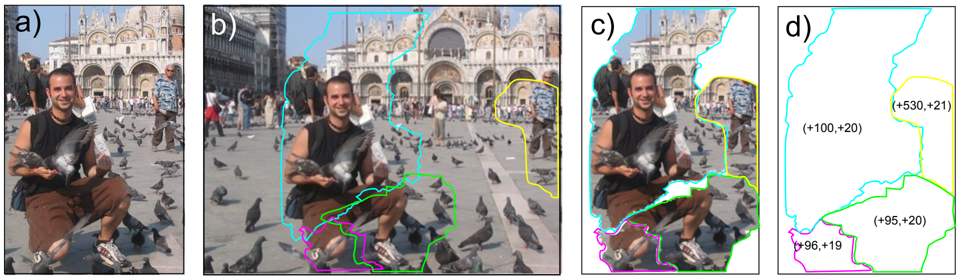

Figure 11.26 Shift maps for image retargeting to reduce width. a) New image $I^{(2)}$ is created from b) by copying parts of the original image $I^{(1)}$ c) These parts are carefully chosen to produce a seamless result. d) The underlying representation is a label at each pixel of the new image that specifies the 2d offset to the position in the original image that will be copied from. An MRF encourages the labels to be piecewise constant and hence the result tends to consist of large chunks copied verbatim.

11.7.2 Optical Flow

To do

11.7.3 Stereo Vision

To do

11.7.4 Rearranging Images

Another application of multi-label Markov random fields is for rearranging images. We are given an original image I_1 and wish to create a new image $I^{(2)}$ by rearranging the pixels from $I^{(1)}$ in some way. Depending on the application we may wish to change the dimensions of the original image (termed *image retargeting*), remove an object or move an object from one place to another.

We will construct a model with hidden variables $\mathbf{y} = \{y_1 \ldots y_N\}$ at each of the N pixels of I^n. Each possible value of $y_n \in \{-N \ldots N\}$ represents a positional offset from this pixel to a different position in image $I^{(1)}$ so that

$$I_n^{(2)} = I_n^{(1)} + y_n. \tag{11.44}$$

The label map $\mathbf{y}$ is hence termed a shift map as it represents a two dimensional offset to the original image. Each possible shiftmap defines an different output image $I^{(2)}$ (figure 11.26).

We model the shiftmap $\mathbf{y}$ as a Markov random field with pairwise costs that encourage smoothness . The result of this is that the only shiftmaps that are piecewise constant have high probability: in other words images where large chunks of the original image are copied verbatim are favoured. We further modify the pairwise costs so that they are lower when adjacent labels that differ encode offsets with similar surrounding regions. This means that where the label does change, it does so in such a way that there is no visible seam in the output image.

The remainder of the model depends on the application (figure **??**:

- To move an object in the image, we specify unary costs at the its position so that the shifts will certainly copy the desired object here. The remainder of the shifts are left free to vary but favour relatively small offsets so that parts of the scene that are far from the change tend to be unperturbed.
- To replace an area of the image, we specify the unary costs so that the remaining part of the image must have a shift of zero (verbatim copying) and the shift in the missing region must be such that it copies from outside this region.
- To retarget and image to larger width, we set the unary costs so that the left and right edges of the new image are forced to have shifts that correspond to the left and right of the original image. We also use the unary costs to specify that vertical shifts must be small.
- To retarget and image to a smaller width, we additionally specify the horizontal offset can only increase. This ensures that new image does not contain replicated objects and that their horizonal order remains constant.

In each case the best solution can be found using the alpha expansion algorithm. Since the labels do not form a metric here, it is necessary to truncate the relevant costs. In practice there are many labels and so a multi-resolution coarse to fine scheme is preferred.

11.8 Appendices

TO DO

- Literature Review
- Pseudocode for Gibbs Sampling
- Pseudocode for binary graph cut solution
- Pseudocode for exact multi-label solution
- Pseudocode for alpha expansion

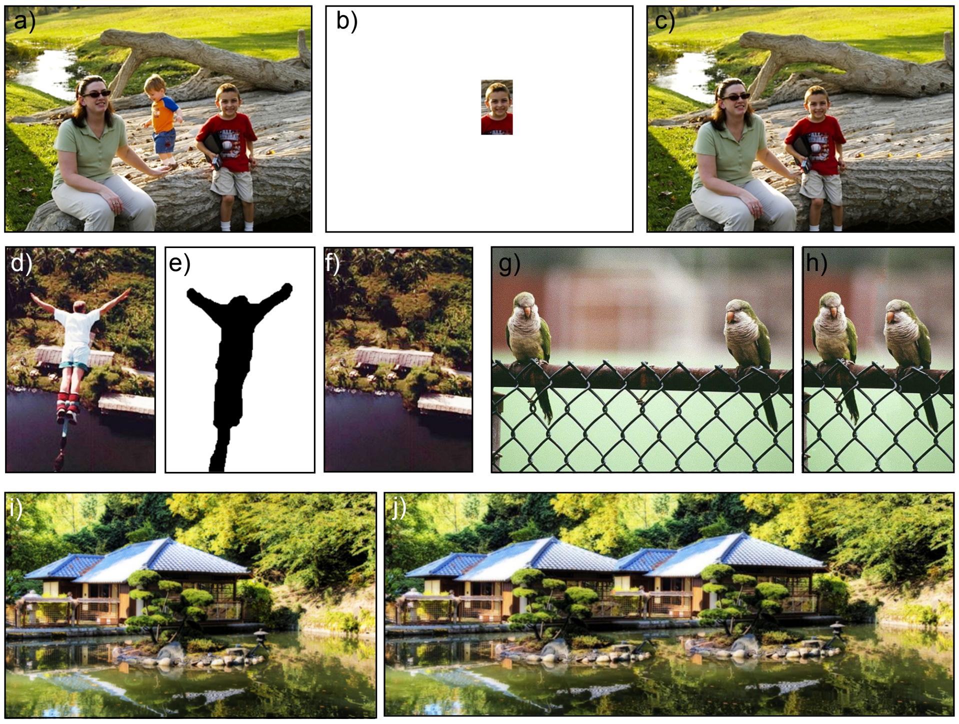

Figure 11.27 Applications of shift maps. Shift maps can be used to take and object from the original image (a), move it to a new position (b) and the fill in the remaining pixels to produce a new picture (c) . They can also be used to remove an undesirable object (d) specified by a mask (e) from an image by filling in the missing area (f). Finally they can to retarget an original image (g) to a smaller size (h), or to retarget an original image (i) bto larger size (j).