Chapter 12

Models for grids

In chapter 11, we discussed models that were structured as chains or trees. In this chapter, we consider models that associate a label with each pixel of an image. Since the unknown quantities are defined on the pixel lattice, models defined on a grid structure are appropriate. In particular, we will consider graphical models in which each label has a direct probabilistic connection to each of its four neighbours. Critically, this means that there are loops in the underlying graphical model and so the dynamic programming and belief propagation approaches of the previous chapter are no longer applicable.

These grid models are predicated on the idea that 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 this chapter, we describe the relative preference for different configurations of labels with a pairwise *Markov random field* or MRF. As we shall see, maximum a posteriori inference for pairwise MRFs is tractable in some circumstances using a family of approaches known collectively as *graph cuts*.

To motivate the grid models, we introduce a representative application. In *image denoising* we observe a corrupted image in which the intensities at a certain proportion of pixels have been randomly changed to another value according to a uniform distribution (figure 12.1). Our goal is to recover the original clean image. We note two important aspects of the problem.

1. Most of the pixels are uncorrupted,so the data usually tell us which intensity value to pick.
2. The uncorrupted image is mainly smooth, with few changes between intensity levels.

Consequently, our strategy will be to construct a generative model where the MAP solution is an image that is mostly the same as the noisy version, but is smoother. As part of this solution, we need to define a probability distribution over images that favors smoothness. In this chapter, we will use a discrete formulation of a Markov random field to fulfil this role.

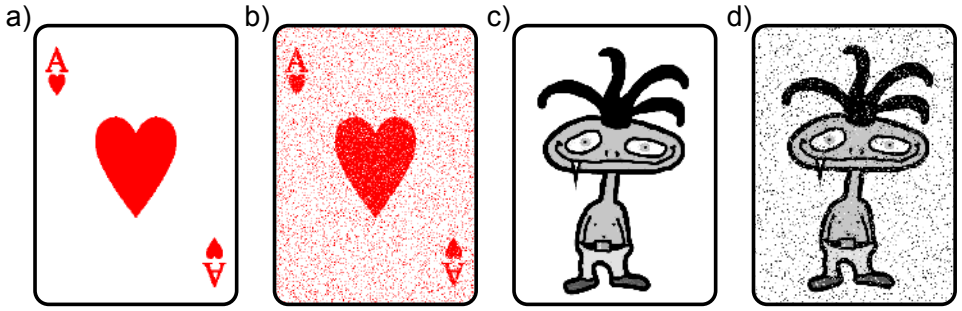

Figure 12.1 Image denoising. a) Original binary image. b) Observed image created by randomly flipping the polarity of a fixed proportion of pixels. Our goal is to recover the original image from the corrupted one. c) Original grayscale image. d) Observed corrupted image is created by setting a certain proportion of the pixels to values drawn from a uniform distribution. Once more, we aim to recover the original image.

12.1 Markov random fields

A *Markov random field* is formally determined by:

- A set of sites $\mathcal{S} = \{1 \ldots N\}$. These will correspond to the N pixel locations.
- A set of random variables $\{w_n\}_{n=1}^N$ associated with each of the sites.
- A set of neighbors $\{\mathcal{N}_n\}_{n=1}^N$ at each of the N sites.

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

$$Pr(w_n|w_{\mathcal{S}\setminus n}) = Pr(w_n|w_{\mathcal{N}_n}). \tag{12.1}$$

In other words, the model should be conditionally independent of all of the other variables given its neighbors. This property should sound familiar: this is exactly how conditional independence works in an undirected graphical model.

Consequently, we can consider a Markov random field (MRF) as an undirected model (section 10.3) that describes the joint probability of the variables as a product of potential functions so that

$$Pr(\mathbf{w}) = \frac{1}{Z}\prod_{j=1}^{J}\phi_j[\mathbf{w}_{\mathcal{C}_j}], \tag{12.2}$$

where $\phi_j[\bullet]$ is the j^{th} potential function and always returns a non-negative value. This value depends on the state of the subset of variables $\mathcal{C}_j \subset \{1, \ldots, N\}$. In this context, this subset is known as a *clique*. The term Z is called the partition function and is a normalizing constant that ensures that the result is a valid probability distribution.

Alternatively, we can rewrite the model as a Gibbs distribution:

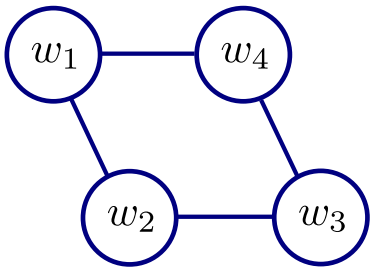

Figure 12.2 Graphical model for worked MRF example. The variables form a 2×2 grid. This is an undirected model where each link represents a potential function defined over the two variables that it connects. Each potential returns a positive number that indicates the tendency of the two variables to take these particular values.

$$Pr(\mathbf{w}) = \frac{1}{Z}\exp\left[-\sum_{j=1}^{J}\psi_j[\mathbf{w}_{\mathcal{C}_j}]\right], \tag{12.3}$$

where $\psi[\bullet] = -\log[\phi[\bullet]]$ is known as a cost function and returns either positive or negative values.

12.1.1 Grid example

In a Markov random field, each potential function $\phi[\bullet]$ (or cost function $\psi[\bullet]$) addresses only a small subset of the variables. In this chapter, we will mainly be concerned with *pairwise Markov random fields* in which the cliques (subsets) consist of only neighboring pairs in a regular grid structure.

To see how the pairwise MRF can be used to encourage smoothness in an image, consider the graphical model for a 2×2 image (figure 12.2). Here, we have defined the probability $Pr(w_{1\ldots4})$ over the associated discrete states as a normalized product of pairwise terms:

$$Pr(\mathbf{w}) = \frac{1}{Z}\phi_{12}(w_1, w_2)\phi_{23}(w_2, w_3)\phi_{34}(w_3, w_4)\phi_{41}(w_4, w_1), \tag{12.4}$$

where $\phi_{mn}(w_m, w_n)$ is a potential function that takes the two states w_m and w_n and returns a positive number.

Let us consider the situation where the world state w_n at each pixel is binary and so takes a value of 0 or 1. The function ϕ_{mn} will now return four possible values depending on which of the four configurations $\{00, 01, 10, 11\}$ of w_m and w_n is present. For simplicity, we will assume that the functions $\phi_{12}, \phi_{23}, \phi_{34}$ and ϕ_{41} are identical and that for each:

$$\begin{aligned} \phi_{mn}(0,0) &= 1.0 \qquad & \phi_{mn}(0,1) &= 0.1 \\ \phi_{mn}(1,0) &= 0.1 \qquad & \phi_{mn}(1,1) &= 1.0. \end{aligned} \tag{12.5}$$

Since there are only four binary states, we can calculate the constant Z explicitly by computing the un-normalized probabilities for each of the 16 possible combinations, and taking the sum. The resulting probabilities for each of the 16 possible states are:

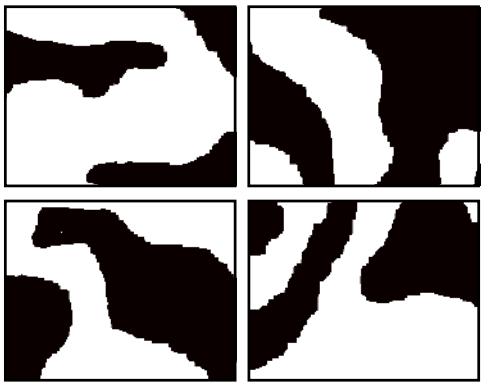

Figure 12.3 Samples from Markov random field prior. Four samples from the MRF prior which were generated using a Gibbs sampling procedure (see section 10.7.2). Each sample is a binary image that is smooth almost everywhere; there are only very occasional changes from black to white and vice-versa. This prior encourages smooth solutions (like the original image in the denoising problems) and discourages isolated changes in label (as are present in the noise).

$w_{1\ldots4}$	$Pr(w_{1\ldots4})$	$w_{1\ldots4}$	$Pr(w_{1\ldots4})$	$w_{1\ldots4}$	$Pr(w_{1\ldots4})$	$w_{1\ldots4}$	$Pr(w_{1\ldots4})$
0000	0.47176	0100	0.00471	1000	0.00471	1100	0.00471
0001	0.00471	0101	0.00005	1001	0.00471	1101	0.00471
0010	0.00471	0110	0.00471	1010	0.00005	1110	0.00471
0011	0.00471	0111	0.00471	1011	0.00471	1111	0.47176

Problem 12.1

The potential functions in equation 12.5 encourage smoothness: the functions ϕ_{mn} return higher values when the neighbors take the same state and lower values when they differ, and this is reflected in the resulting probabilities.

We can visualize this by scaling this model up to a larger image-sized grid where there is one node per pixel, and drawing samples from the resulting probability distribution (figure 12.3). The resulting binary images are mostly smooth, with only occasional changes between the two values.

It should be noted that for this more realistically-sized model, we cannot compute the normalizing constant Z by brute force as for the 2×2 case. For example, with 10,000 pixels each taking a binary value, the normalizing constant is the sum of $2^{10,000}$ terms. In general we will have to cope with only knowing the probabilities up to an unknown scaling factor.

12.1.2 Image denoising with discrete pairwise MRFs

Now we will apply the pairwise Markov random field model to the denoising task. Our goal is to recover the original image pixel values from the observed noisy image.

More precisely, the observed image $\mathbf{x} = \{x_1, x_2, \ldots, x_N\}$ is assumed to consist of discrete variables where the different possible values (labels) represent different intensities. Our goal is to recover the original uncorrupted image $\mathbf{w} = \{w_1, w_2, \ldots, w_N\}$, 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 $\mathbf{w}$ using Bayes' rule

$$Pr(w_{1\ldots N}|x_{1\ldots N}) = \frac{\prod_{n=1}^{N} Pr(x_n|w_n) Pr(w_{1\ldots N})}{Pr(x_{1\ldots N})}, \tag{12.6}$$

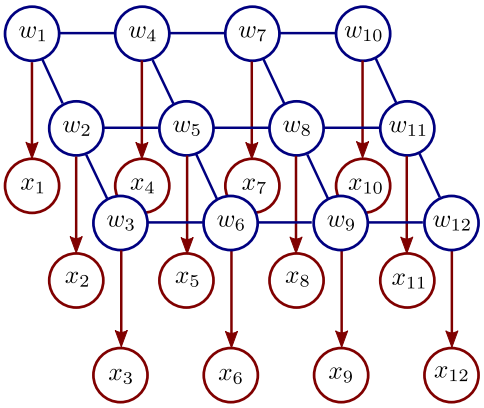

Figure 12.4 Denoising model. The observed data $\mathbf{x}_n$ at pixel n is conditionally dependent on the associated world state w_n (red directed edges). Each world state w_n has undirected edges to its four-connected neighbors (blue undirected edges). This is hence a mixed model: it contains both directed and undirected elements. Together the world states are connected in a Markov random field with cliques that consist of neighboring pairs of variables. For example variable w_5 contributes to cliques $\mathcal{C}_{25}, \mathcal{C}_{45}, \mathcal{C}_{65}, \mathcal{C}_{85}$.

where we have assumed that the conditional probability $Pr(x_{1\ldots N}|w_{1\ldots N})$ factorizes into a product of individual terms associated with each pixel. We will first consider denoising binary images in which the noise process flips the pixel polarity with probability ρ so that

$$\begin{aligned} Pr(x_n|w_n = 0) &= \text{Bern}_{x_n}[\rho] \\ Pr(x_n|w_n = 1) &= \text{Bern}_{x_n}[1-\rho]. \end{aligned} \tag{12.7}$$

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

We now define a prior that encourages the labels w_n to be smooth: we want them to mostly agree with the observed image, but to discourage configurations with isolated changes in label. To this end, we model the prior as a pairwise MRF. Each pair of four-connected neighboring pixels contributes one clique so that

$$Pr(w_{1\ldots N}) = \frac{1}{Z}\exp\left[-\sum_{(m,n)\in\mathcal{C}}\psi[w_m, w_n, \boldsymbol{\theta}]\right], \tag{12.8}$$

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

$$\psi[w_m = j, w_n = k, \boldsymbol{\theta}] = \theta_{jk}, \tag{12.9}$$

so when the first variable w_m in the clique takes label j and the second variable w_n takes label k ,we pay a price of θ_{jk}. As before, 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 associated graphical model is illustrated in figure 12.4. It is a mixed model, containing both directed and undirected links. The likelihood terms (equation 12.7) contribute the red directed links between the observed data and the denoised image

at each pixel, and the MRF prior (equation 12.8) contributes the blue grid that connects the pixels together.

12.2 MAP inference for binary pairwise MRFs

To denoise the image, we estimate the variables $\{w_n\}_{n=1}^N$ using MAP inference; we aim to find the set of world states $\{w_n\}_{n=1}^N$ that maximizes the posterior probability $Pr(w_{1\ldots N}|\mathbf{x}_{1\ldots N})$ so that

$$\begin{aligned}\hat{w}_{1\ldots N} &= \underset{w_{1\ldots N}}{\operatorname{argmax}}\left[Pr(w_{1\ldots N}|\mathbf{x}_{1\ldots N})\right] \\ &= \underset{w_{1\ldots N}}{\operatorname{argmax}}\left[\prod_{n=1}^{N} Pr(x_n|w_n)Pr(w_{1\ldots N})\right] \\ &= \underset{w_{1\ldots N}}{\operatorname{argmax}}\left[\sum_{n=1}^{N} \log[Pr(x_n|w_n)] + \log[Pr(w_{1\ldots N})]\right], \end{aligned} \tag{12.10}$$

where we have applied Bayes' rule and transformed to the log domain. Because the prior is an MRF with pairwise connections, we can express this as

$$\begin{aligned}\hat{w}_{1\ldots N} &= \underset{w_{1\ldots N}}{\operatorname{argmax}}\left[\sum_{n=1}^{N} \log[Pr(x_n|w_n)] - \sum_{(m,n)\in\mathcal{C}} \psi[w_m, w_n, \boldsymbol{\theta}]\right] \\ &= \underset{w_{1\ldots N}}{\operatorname{argmin}}\left[\sum_{n=1}^{N} -\log[Pr(x_n|w_n)] + \sum_{(m,n)\in\mathcal{C}} \psi[w_m, w_n, \boldsymbol{\theta}]\right] \\ &= \underset{w_{1\ldots N}}{\operatorname{argmin}}\left[\sum_{n=1}^{N} U_n(w_n) + \sum_{(m,n)\in\mathcal{C}} P_{mn}(w_m, w_n)\right], \end{aligned} \tag{12.11}$$

where $U_n(w_n)$ denotes the *unary* term at pixel n. This is a cost for observing the data at pixel n given state w_n, and is the negative log likelihood term. Similarly, $P_{mn}(w_m, w_n)$ denotes the *pairwise* term. This is a cost for placing labels w_m and w_n at neighboring locations m and n, and is due to the clique costs $\psi[w_m, w_n, \boldsymbol{\theta}]$ from the MRF prior. Note that we have omitted the term $-\log[Z]$ from the MRF definition as it is constant with respect to the states $\{w_n\}_{n=1}^N$ and hence does not affect the optimal solution.

The cost function in equation 12.11 can be optimized using a set of techniques known collectively as *graph cuts*. We will consider three cases:

- binary MRFs (i.e., $w_i \in \{0, 1\}$) where the costs for different combinations of adjacent labels are "submodular" (we will explain what this means later in the chapter). Exact MAP inference is tractable here.

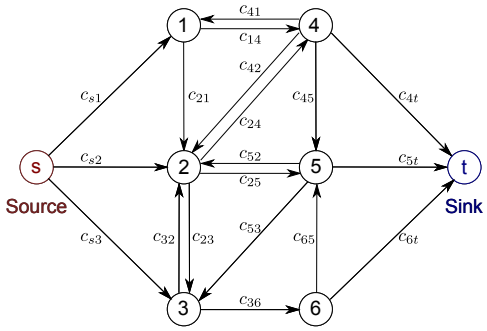

Figure 12.5 Max flow problem: we are given a network of vertices connected by directed edges, each of which has a non-negative capacity c_{mn}. There are two special vertices s and t termed the source and sink, respectively. In the max-flow problem, we seek to push as much 'flow' from source to sink while respecting the capacities of the edges.

- multi-label MRFs (i.e., $w_i \in \{1, 2, \ldots, K\}$) 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.

To solve these MAP inference tasks, we will translate them into the form of *maximum flow* (or *max-flow*) problems. 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 subsequent parts of the chapter, we describe how to translate MAP inference in Markov random fields into a max-flow problem.

12.2.1 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 12.5). 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 – the so-called *max-flow* solution – 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 separate the source from the sink. In other words, when these edges are removed, there is no path from the source to the sink. More precisely, a cut partitions the vertices into two groups: vertices that can be reached by some path from the source, but cannot reach the sink, and vertices that cannot be reached from the source, but can reach the sink via some path. For short, we will refer to 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.

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 12.6).

Consider choosing any path from the source to the 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, 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 maximum flow, and the saturated edges form the minimum cut.

In the full algorithm, 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.

12.2.2 MAP inference: binary variables

Algorithm 12.1

Recall that to find the MAP solution we must find

$$\hat{w}_{1\ldots N} = \underset{w_{1\ldots N}}{\operatorname{argmin}} \left[\sum_{n=1}^{N} U_n(w_n) + \sum_{(m,n)\in\mathcal{C}} P_{mn}(w_m, w_n) \right], \tag{12.12}$$

where $U_n(w_n)$ denotes the *unary* term and $P_{mn}(w_m, w_n)$ 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

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

where $\theta_{01}, \theta_{10} > 0$. We discuss the more general case later in this section.

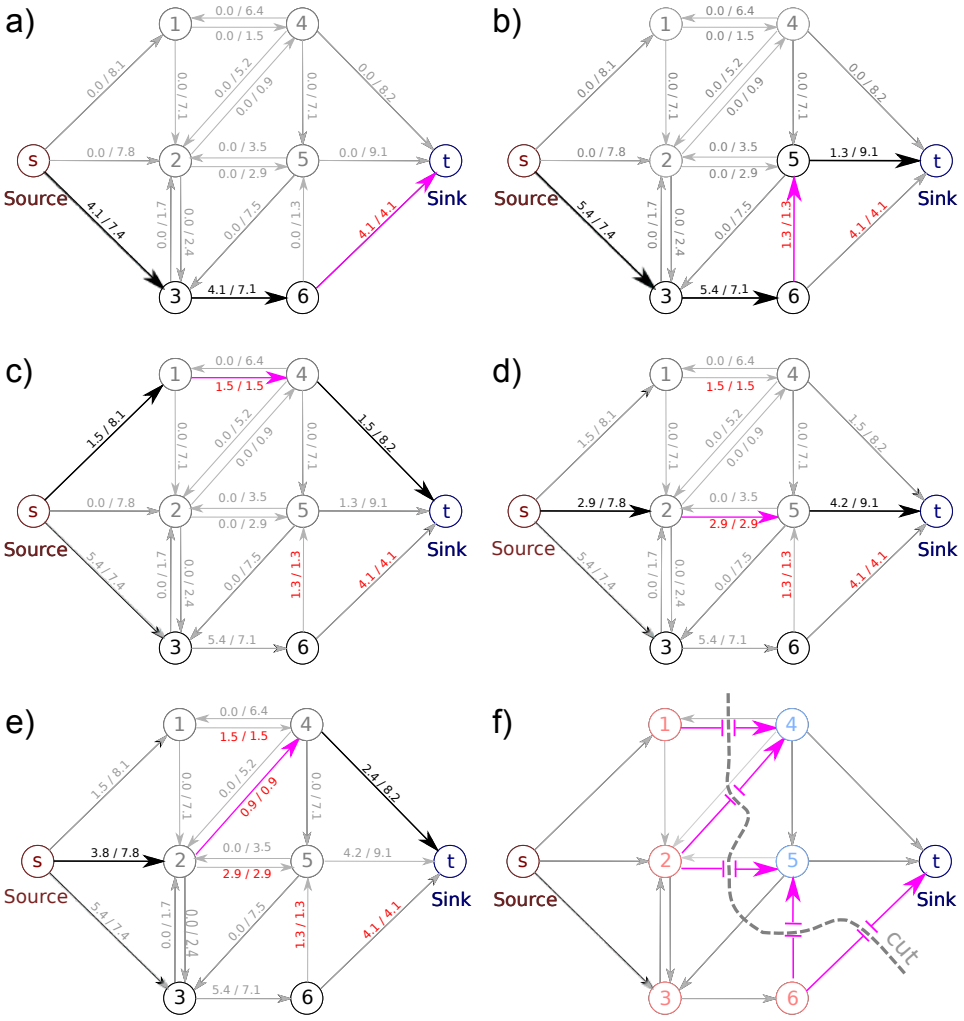

Figure 12.6 Augmenting paths algorithm for max-flow. The numbers attached to the edges correspond to current flow / capacity. a) We choose any path from source to sink with spare capacity, and push as much flow as possible along this path. The edge with the smallest capacity (here edge 6-t) saturates. b) We then choose another path where there is still spare capacity and push as much flow as possible. Now edge 6-5 saturates. c-e) We repeat this until there is no path from source to sink that does not contain a saturated edge. The total flow pushed is the maximum flow. f) 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 line) consists of the saturated edges in the max-flow problem. In this example, the paths were chosen arbitrarily, but to ensure that this algorithm converges in the general case, we should choose the remaining path with the greatest capacity at each step.

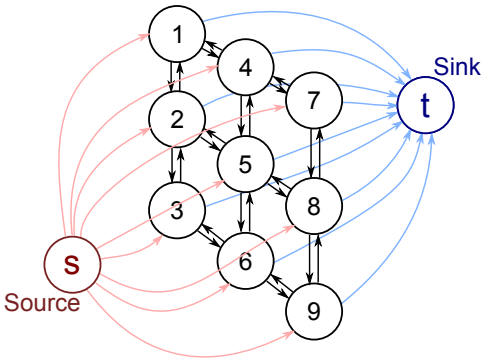

Figure 12.7 Graph structure for finding MAP solution for a MRF with binary labels and pairwise connections in a 3×3 image. There is one vertex per pixel and neighbors in the pixel grid are connected by reciprocal pairs of directed edges. Each pixel vertex receives a connection from the source and sends a connection 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.

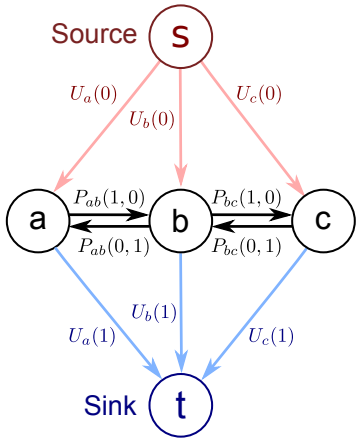

Figure 12.8 Graph construction for binary MRF with diagonal pairwise terms using simple 1D example. After the cut, vertices attached to the source are given label 1 and vertices attached to the sink are given label 0. We hence attach the appropriate unary costs to the links between the sink/source and the pixel vertices. The pairwise costs are attached to the horizontal links between pixels as shown. This arrangement ensures that the correct cost is paid for each of the eight possible solutions (see figure 12.9).

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 adjacent vertices in the pixel grid. In addition, there is a directed edge from the source to every vertex and a directed edge from every vertex to the sink (figure 12.7).

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 (assuming the general case where the two edges have different capacities) – this is unnecessary and will inevitably incur a greater cost than cutting one or the other. We will label pixels where the edge to the source was cut as $w_n = 0$, and pixels where the edge to the sink was cut as having label $w_n = 1$. So each plausible minimum cut is associated with a pixel labeling.

Our goal is now to assign capacities to the edges, so the cost of each cut matches the cost of the associated labeling as prescribed in equation 12.12. For simplicity, we illustrate this with a 1D image with three pixels (figure 12.8), but we stress that all the ideas are also valid for 2D images and higher dimensional constructions.

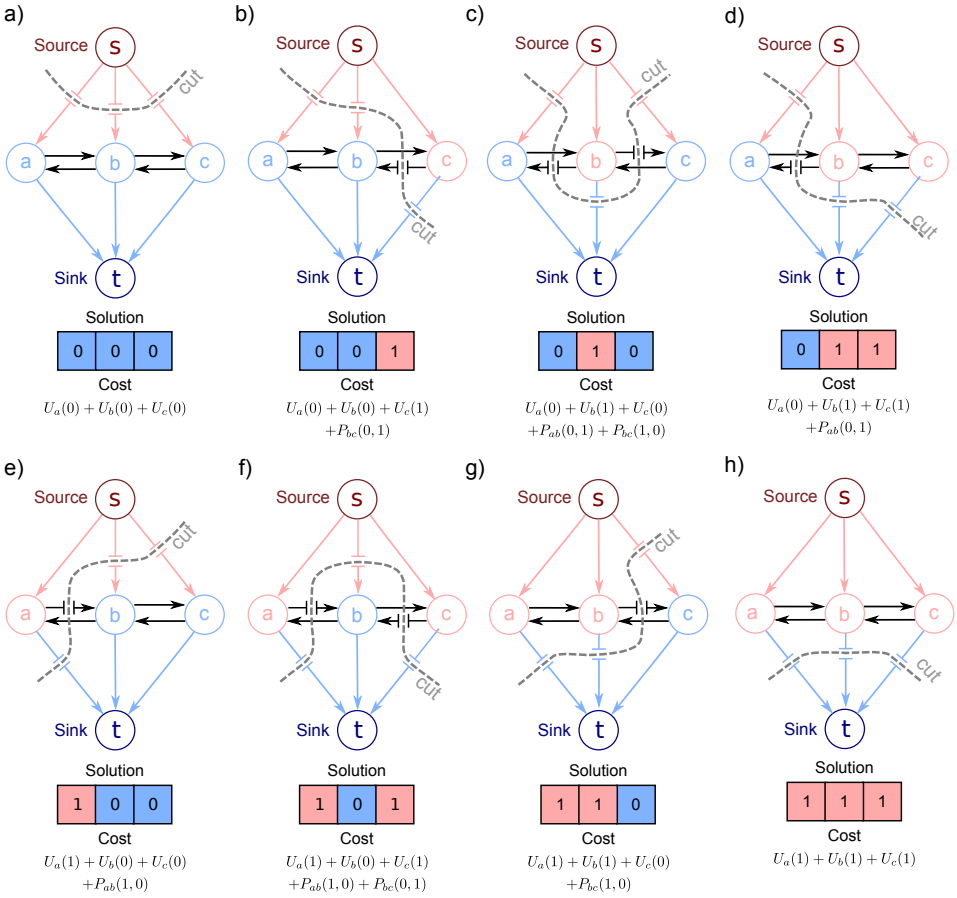

Figure 12.9 Eight possible solutions for three pixel example. When we set the costs as in figure 12.8, each solution has the appropriate cost. a) For example, the solution $(a=0, b=0, c=0)$ requires us to cut edges $s{-}a, s{-}b, s{-}c$ and pay the cost $U_a(0)+U_b(0)+U_c(0)$. b) For the solution $(a=0, b=0, c=1)$ we must cut edges $s{-}a, s{-}b, c{-}t$, and $c{-}b$ (to prevent flow through the path $s{-}c{-}b{-}t$). This incurs a total cost of $U_a(0)+U_b(0)+U_c(1)+P_{bc}(0,1)$. c) Similarly, in this example with $(a=0, b=1, c=0)$, we pay the appropriate cost $U_a(0)+U_b(1)+U_c(0)+P_{ab}(0,1)+P_{bc}(1,0)$. d-h) The other five possible configurations.

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 from the source to a given pixel (and hence assign $w_n=0$), we pay the cost $U_n(0)$. Conversely, if we cut the edge from the pixel to the sink (and hence assign $w_n=1$), we pay the cost $U_n(1)$.

We attach the pairwise costs $P_{mn}(1,0)$ and $P_{mn}(0,1)$ to the pairs of edges between 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 sepa-

Problem 12.2

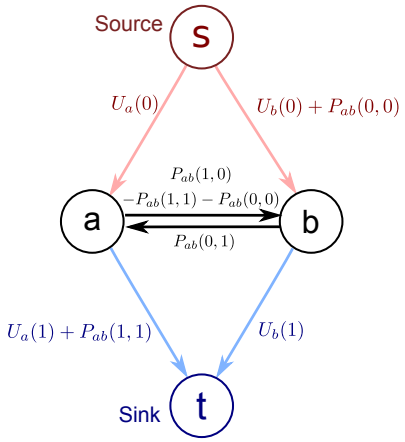

Figure 12.10 Graph structure for general (i.e., non-diagonal) pairwise costs. Consider the solution $(a = 0, b = 0)$. We must break the edges $s-a$ and $s-b$ giving a total cost of $U_a(0) + U_b(0) + P_{ab}(0,0)$. For the solution $(a = 1, b = 0)$ we must break the edges $a-t, a-b$ and $s-b$ giving a 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.

rate source from sink. The cuts corresponding to all eight possible configurations of the three pixel model and their costs are illustrated in figure 12.9.

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 12.12. It follows that the minimum cut on this graph will have the minimum cost and the associated labeling $\hat{w}_{1\ldots N}$ will correspond to the maximum a posteriori solution.

General pairwise costs

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

$$\begin{aligned} P_{mn}(0,0) &= \theta_{00} \qquad & P_{mn}(1,0) &= \theta_{10} \\ P_{mn}(0,1) &= \theta_{01} \qquad & P_{mn}(1,1) &= \theta_{11}. \end{aligned} \tag{12.13}$$

Problem 12.3

To illustrate this, we use an even simpler graph with only two pixels (figure 12.10). 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 the configuration where $w_a = 0$ and $w_b = 0$. Unfortunately, we would also pay it in the case where $w_a = 1$ and $w_b = 0$. Hence, we subtract the same cost from the edge $a-b$, 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 $a-b$. In this way, we associate the correct costs with each labeling.

Reparameterization

Algorithm 12.2

The preceding discussion assumed that the edge costs are all non-negative and can be valid capacities in the max-flow problem. If they are not, then it is not possible to compute the MAP solution. Unfortunately, it is often the case that they are negative; even if the original unary and pairwise terms were positive, the edge $a-b$ in figure 12.10 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 adjust 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 labeling will be unchanged.

We consider two reparameterizations (figure 12.11). 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 all non-negative.

A more subtle type of reparameterization is illustrated in figure 12.11c. By changing the costs in this way, we increase the total cost of each possible solution by β. For example, in the assignment $(w_a = 0, w_b = 1)$ we must cut the links $s-a$, $b-a$ and $b-t$ giving a total cost of $U_a(0) + U_b(1) + P_{ab}(0,1) + \beta$.

Problem 12.4

Applying the reparameterization in figure 12.11c to the general construction in figure 12.10, we must ensure that the capacities on edges between pixel nodes are non-negative so that

$$\begin{aligned} \theta_{10} - \theta_{11} - \theta_{00} - \beta &\geq 0 \qquad (12.14) \\ \theta_{01} + \beta &\geq 0. \qquad (12.15) \end{aligned}$$

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

$$\theta_{01} + \theta_{10} - \theta_{11} - \theta_{00} \geq 0. \qquad (12.16)$$

If this condition holds, the problem is termed *submodular* , and the graph can be reparameterized to have only non-negative costs. It can then 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, the former case is common for vision problems; we generally favor smooth solutions where neighboring 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 12.12 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 for adjacent labels being different are the same ($\theta_{01} = \theta_{10}$) and that there is no cost when neighboring labels are the same ($\theta_{00}, \theta_{11} = 0$); 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 center 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 merely determine the polarity.

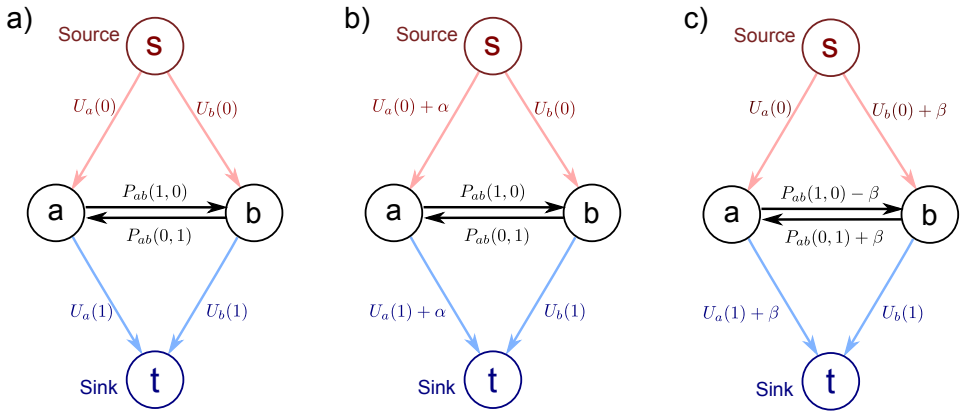

Figure 12.11 Reparameterization. a) Original graph construction. b) Reparameterization 1. Adding a constant cost α to the connections from a pixel vertex to both the source and sink results in a problem with the same MAP solution. Since we must cut either, but not both of these edges, every solution 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.

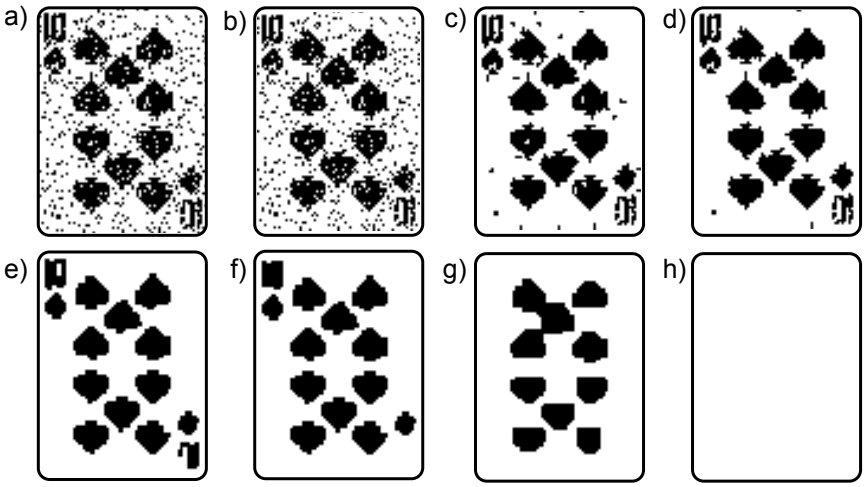

Figure 12.12 Denoising results. a) Observed noisy image. b-h) Maximum a posteriori solution as we increase zero-diagonal pairwise costs. When the pairwise costs are low, the unary terms dominate and the MAP solution is the same as the observed image. As the pairwise costs increase the image gets more and more smooth until eventually it becomes uniform.

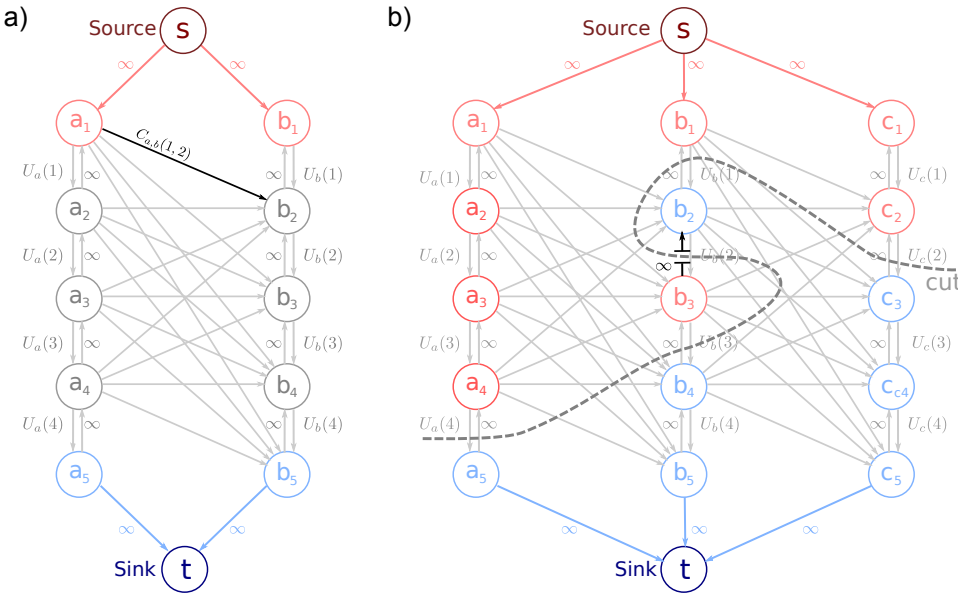

Figure 12.13 a) Graph setup for multi-label case for two pixels (a, b) and four labels $(1, 2, 3, 4)$. There is a chain of five vertices associated with each pixel. The four vertical edges between these vertices are assigned the unary costs for the four labels. The minimum cut must break this chain to 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 example with three pixels. Here, the chain of vertices associated with the central pixel is cut in more than one place and so the labeling has no clear interpretation. However, for this to happen, a constraint link must be cut, and hence this solution has an infinite cost.

12.3 MAP inference for multi-label pairwise MRFs

We now investigate MAP inference using MRF priors with pairwise connections when the world state w_n at each pixel can take multiple labels $\{1, 2, \ldots, K\}$. To solve the multi-label problem, we change the graph construction (figure 12.13a). With K labels and N pixels, we introduce $(K+1)N$ vertices into the graph.

Algorithm 12.3

For each pixel, the $K + 1$ associated vertices are stacked. The top and bottom of the stack are connected to the source and sink by edges with infinite capacity. 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 k^{th} edge in this chain as indicating that the pixel takes label k and this incurs the appropriate cost of $U_n(k)$.

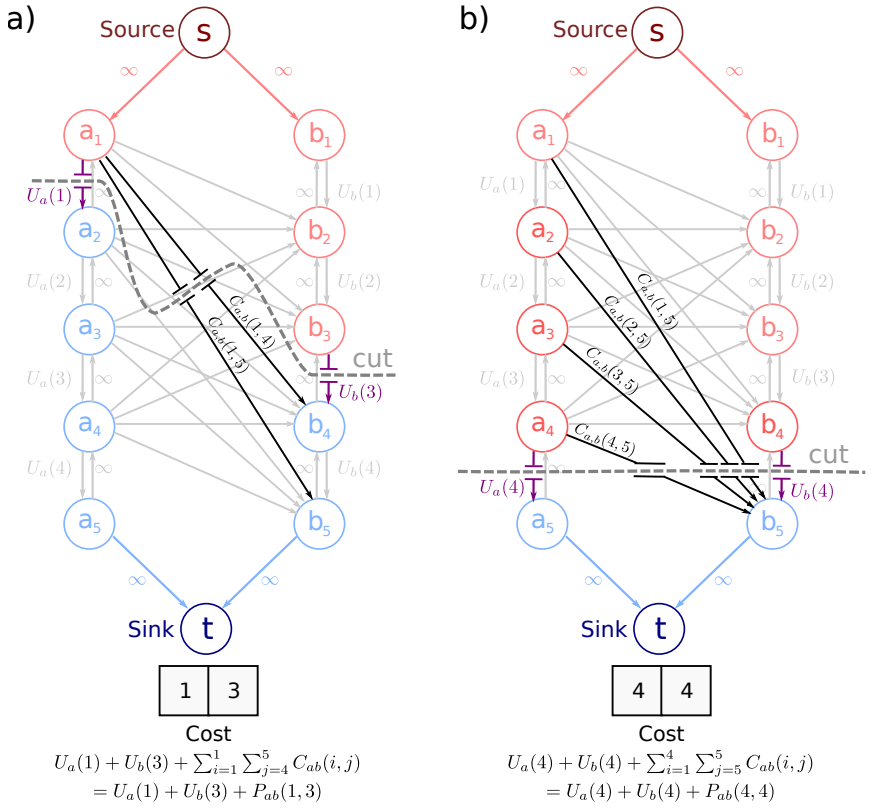

Figure 12.14 Example cuts for multi-label case. To separate the source and sink, we must cut all of the links that pass from above the chosen label for pixel a to below the chosen label for pixel b. a) Pixel a is set to label 1 and pixel b is set to label 3 meaning we must cut the links from vertex a_1 to nodes b_4 and b_5. b) Pixel a takes label 4 and pixel b takes label 4.

Problem 12.5
Problem 12.6
Problem ??

To ensure that only a single edge from the chain is part of the minimum cut (and hence that each cut corresponds to one valid labeling), we add *constraint edges*. These are edges of infinite capacity that are strategically placed to prevent certain cuts occurring. In this case, the constraint edges 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 12.13b).

In figure 12.13a, there are also diagonal inter-pixel edges from the vertices associated with pixel a to those 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 edge costs to be

$$C_{ab}(i,j) = P_{ab}(i,j-1) + P_{ab}(i-1,j) - P_{ab}(i,j) - P_{ab}(i-1,j-1), \quad (12.17)$$

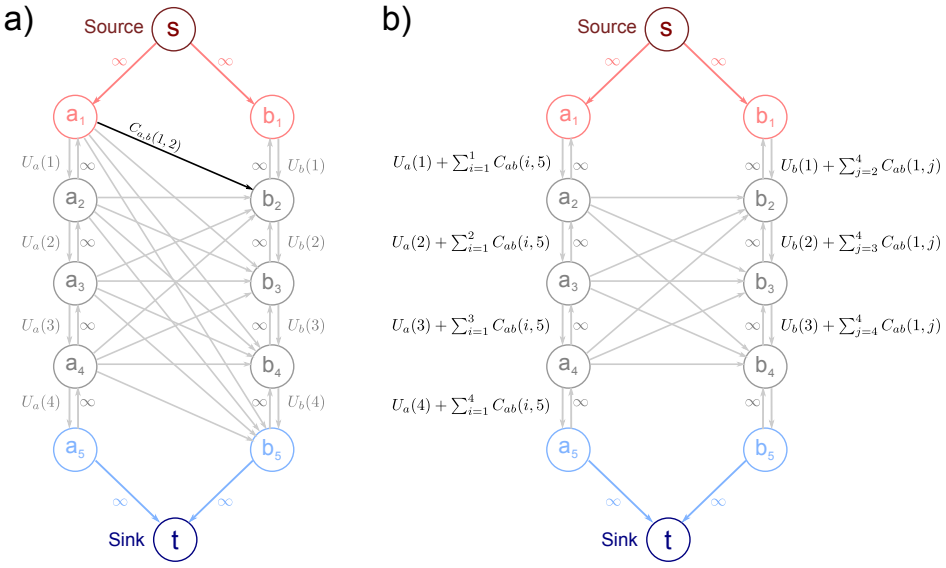

Figure 12.15 Reparameterization for multi-label graph cuts. The original construction (a) is equivalent to construction (b). The label at pixel b determines which edges that leave node a_1 are cut. Hence, we can remove these edges and add the extra costs to the vertical links associated with pixel b. 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 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.

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

$$\begin{aligned} P_{ab}(i,0) &= 0 \qquad P_{ab}(i,K+1) = 0 \qquad \forall\ i \in \{0\ldots K+1\} \\ P_{ab}(0,j) &= 0 \qquad P_{ab}(K+1,j) = 0 \qquad \forall\ j \in \{0\ldots K+1\}. \end{aligned} \tag{12.18}$$

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 12.14). So, the total cost due to the inter-pixel edges for assigning label I to pixel a and label J to pixel b is

Problem 12.7

$$\begin{aligned} \sum_{i=1}^{I}\sum_{j=J+1}^{K+1} C_{ab}(i,j) &= \sum_{i=1}^{I}\sum_{j=J+1}^{K+1} P_{ab}(i,j-1)+P_{ab}(i-1,j)-P_{ab}(i,j)-P_{ab}(i-1,j-1) \\ &= P_{ab}(I,J)+P_{ab}(0,J)-P_{ab}(I,K+1)-P_{ab}(0,K+1) \\ &= P_{ab}(I,J). \end{aligned} \tag{12.19}$$

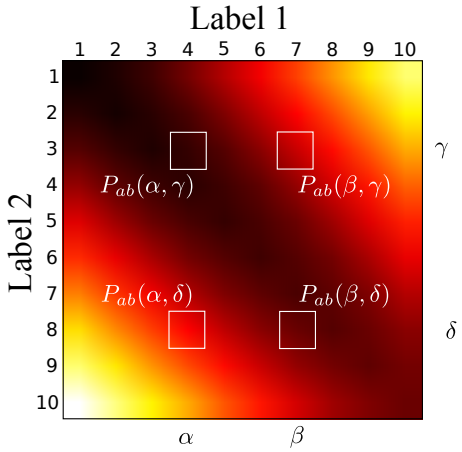

Figure 12.16 Submodularity constraint for multi-label case. Color at position (m,n) indicates pairwise costs $P_{ab}(m,n)$. For all edges in the graph to be positive, we require that the pairwise terms obey $P_{ab}(\beta,\gamma) + P_{ab}(\alpha,\delta) - P_{ab}(\beta,\delta) - P_{ab}(\alpha,\gamma) \geq 0$ for all $\alpha,\beta,\gamma,\delta$ such that $\beta > \alpha$ and $\delta > \gamma$. In other words, for any four positions arranged in a square configuration as in the figure, the sum of the two costs on the diagonal from top-left to bottom-right must be less than the sum on the off diagonal. If this condition holds, the problem can be solved in polynomial time.

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) edge terms have negative costs, it is possible to reparameterize 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+1} by adding terms to the intra-pixel edges associated with the unary terms (figure 12.15). These intra-pixel edges can then be reparameterized as described above if necessary. Unfortunately, we can neither remove nor reparameterize the remaining inter-pixel edges, so we require that

$$C_{ab}(i,j) = P_{ab}(i,j-1) + P_{ab}(i-1,j) - P_{ab}(i,j) - P_{ab}(i-1,j-1) \geq 0. \quad (12.20)$$

By mathematical induction, we get the more general result (figure 12.16),

$$P_{ab}(\beta,\gamma) + P_{ab}(\alpha,\delta) - P_{ab}(\beta,\delta) - P_{ab}(\alpha,\gamma) \geq 0, \quad (12.21)$$

where $\alpha,\beta,\gamma,\delta$ are any four values of the state y such that $\beta > \alpha$ and $\delta > \gamma$. This is the multi-label generalization of the submodularity condition (equation 12.16). An important class of pairwise costs that are submodular are those that are convex in the absolute difference $|w_i - w_j|$ between the labels at adjacent pixels (figure 12.17a). Here, smoothness is encouraged; the penalty becomes increasingly stringent as the jumps between labels increase.

12.4 Multi-label MRFs with non-convex potentials

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

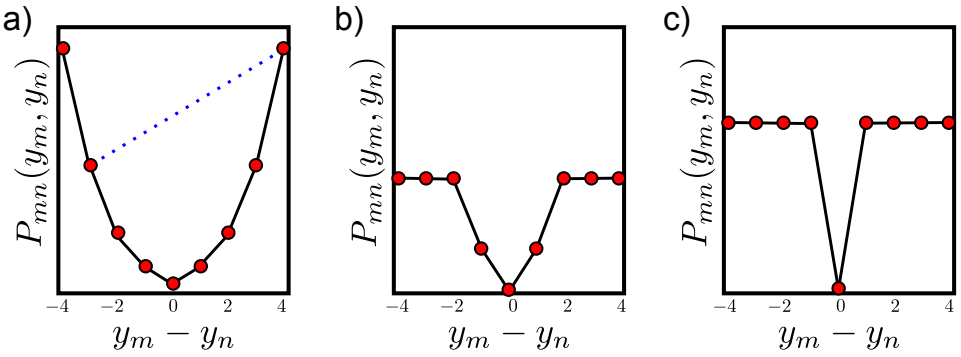

Figure 12.17 Convex vs. non-convex potentials. The method for MAP inference for multi-valued variables depends on whether the costs are a convex or non-convex function of the difference in labels. a) Quadratic function (convex), $P_{mn}(w_m, w_n) = \kappa(w_m - w_n)^2$. For convex functions, it is possible to draw a chord between any two points on the function without intersecting the function elsewhere (e.g., dotted blue line). b) Truncated quadratic function (non-convex), $P_{mn}(w_m, w_n) = \min(\kappa_1, \kappa_2(w_m - w_n)^2)$. c) Potts model (non-convex), $P_{mn}(w_m, w_n) = \kappa(1 - \delta(w_m - w_n))$.

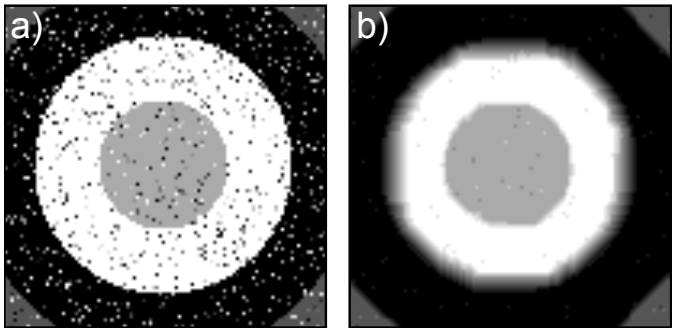

Figure 12.18 Denoising results with convex (quadratic) pairwise costs. a) Noisy observed image. b) Denoised image has artiWfacts where there are large intensity changes in the original image. Convex costs imply that there is a lower cost for a number of small changes rather than a single large one.

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 12.18).

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

Problem 12.8

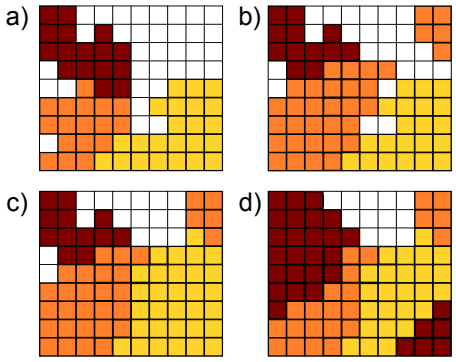

Figure 12.19 The alpha-expansion algorithm breaks the problem down into a series of binary sub-problems. At each step, we choose a label α and we *expand*: for each pixel we either leave the label as it is, or replace it with α. This sub-problem is solved in such a way that it is guaranteed to decrease the multilabel cost function. a) Initial labeling. b) Orange label is expanded: each label stays the same or becomes orange. c) Yellow label is expanded. d) Red label is expanded.

Potts model (figures 12.17b-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 12.21). Here, the MAP solution cannot in general be found exactly with the method described previously, and the problem is NP-hard. Fortunately, there are good approximate methods for optimizing such problems, one of which is the alpha-expansion algorithm.

12.4.1 Inference: alpha-expansion

Algorithm 12.4

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 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 space occupied by label α in the solution expands at each iteration (figure 12.19). 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

$$\begin{aligned} P(\alpha,\beta) &= 0 \quad \Leftrightarrow \alpha = \beta \\ P(\alpha,\beta) &= P(\beta,\alpha) \geq 0 \\ P(\alpha,\beta) &\leq P(\alpha,\gamma) + P(\gamma,\beta). \end{aligned} \tag{12.22}$$

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

In the alpha-expansion graph construction (figure 12.20), 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

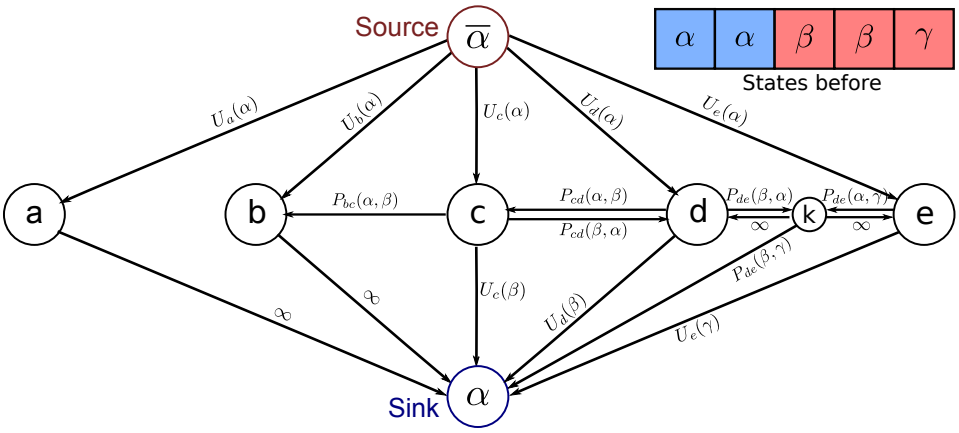

Figure 12.20 Alpha-expansion graph setup. Each pixel node (a,b,c,d,e) is connected to the source and the sink by edges with costs $U_\bullet(\overline{\alpha})$ and $U_\bullet(\alpha)$, respectively. In the minimum cut, exactly one of these links will be cut. The nodes and vertices describing the relationship between neighboring pixels depend 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.

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 current 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 12.20 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 pairwise cost or $\alpha-\beta$ with pairwise cost $P_{ij}(\alpha,\beta)$. Here we add a single edge connecting pixel j to pixel i with pairwise cost $P_{ij}(\alpha,\beta)$. Pixels b and c in figure 12.20 have this relationship.

- Both pixels i and j have the same label β. Here the final solution may be α–α with zero pairwise cost, $\beta-\beta$ with zero pairwise cost, $\alpha-\beta$ with pairwise cost $P_{ij}(\alpha,\beta)$ or $\beta-\alpha$ with pairwise cost $P_{ij}(\beta,\alpha)$. We add two edges between the pixels representing the two non-zero pairwise costs. Pixels c and d in figure 12.20 have this relationship.

- Pixel i has label β and pixel j has a second label γ. Here the final solution may be $\alpha-\alpha$ with zero pairwise cost, $\beta-\gamma$ with pairwise cost $P_{ij}(\beta,\gamma)$, $\beta-\alpha$

with pairwise cost $P_{ij}(\beta,\alpha)$, or $\alpha-\gamma$ with pairwise cost $P_{ij}(\alpha,\gamma)$. We add a new vertex k between vertices i and j, and add the three non-zero pairwise costs to edges $k-\alpha$, $i-k$, and $j-k$, respectively. Pixels d and e in figure 12.20 have this relationship.

Three example cuts are shown in figure 12.21.

Note that this construction critically relies on the triangle inequality (equation 12.22). For example, consider pixels d and e in figure 12.21a. If the triangle inequality does not hold so that $P_{de}(\beta,\gamma) > P_{de}(\beta,\alpha) + P_{de}(\alpha,\gamma)$, then the wrong costs will be assigned; rather than the link $k-\alpha$, the two links $d-k$ and $e-k$ will both be cut, and the wrong cost will be assigned. 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 (sum of the unary and pairwise costs) can be computed for the new label map and the answer accepted if the cost has decreased.

Problem 12.9

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 12.22 shows an example of multi-label denoising using the alpha-expansion algorithm. On each iteration, one of the labels is chosen and expands, 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.

12.5 Conditional random fields

In the models presented in this chapter, the Markov random fields have described the prior $Pr(\mathbf{w})$ in a generative model of the image data. We could alternatively describe the joint probability distribution $Pr(\mathbf{w},\mathbf{x})$ with the undirected model

$$Pr(\mathbf{w},\mathbf{x}) = \frac{1}{Z}\exp\left[-\sum_{c}\psi_C[\mathbf{w}] - \sum_{d}\zeta[\mathbf{w},\mathbf{x}]\right], \tag{12.23}$$

where the functions $\psi[\bullet]$ encourage certain configurations of the label field, and the functions $\zeta[\bullet,\bullet]$ encourage agreement between the data and the label field. If we now condition on the data (i.e., assume that it is fixed), then we can use the relation $Pr(\mathbf{w}|\mathbf{x}) \propto Pr(\mathbf{w},\mathbf{x})$ to write

$$Pr(\mathbf{w}|\mathbf{x}) = \frac{1}{Z_2}\exp\left[-\sum_{c}\psi_C[\mathbf{w}] - \sum_{d}\zeta[\mathbf{w},\mathbf{x}]\right], \tag{12.24}$$

where $Z_2 = Z\cdot Pr(\mathbf{x})$. This discriminative model is known as a *conditional random field* or *CRF*.

We can choose the functions $\zeta[\bullet,\bullet]$ so that they each determine the compatibility of one label w_n to its associated measurement x_n. If the functions $\psi[\bullet]$ are used to

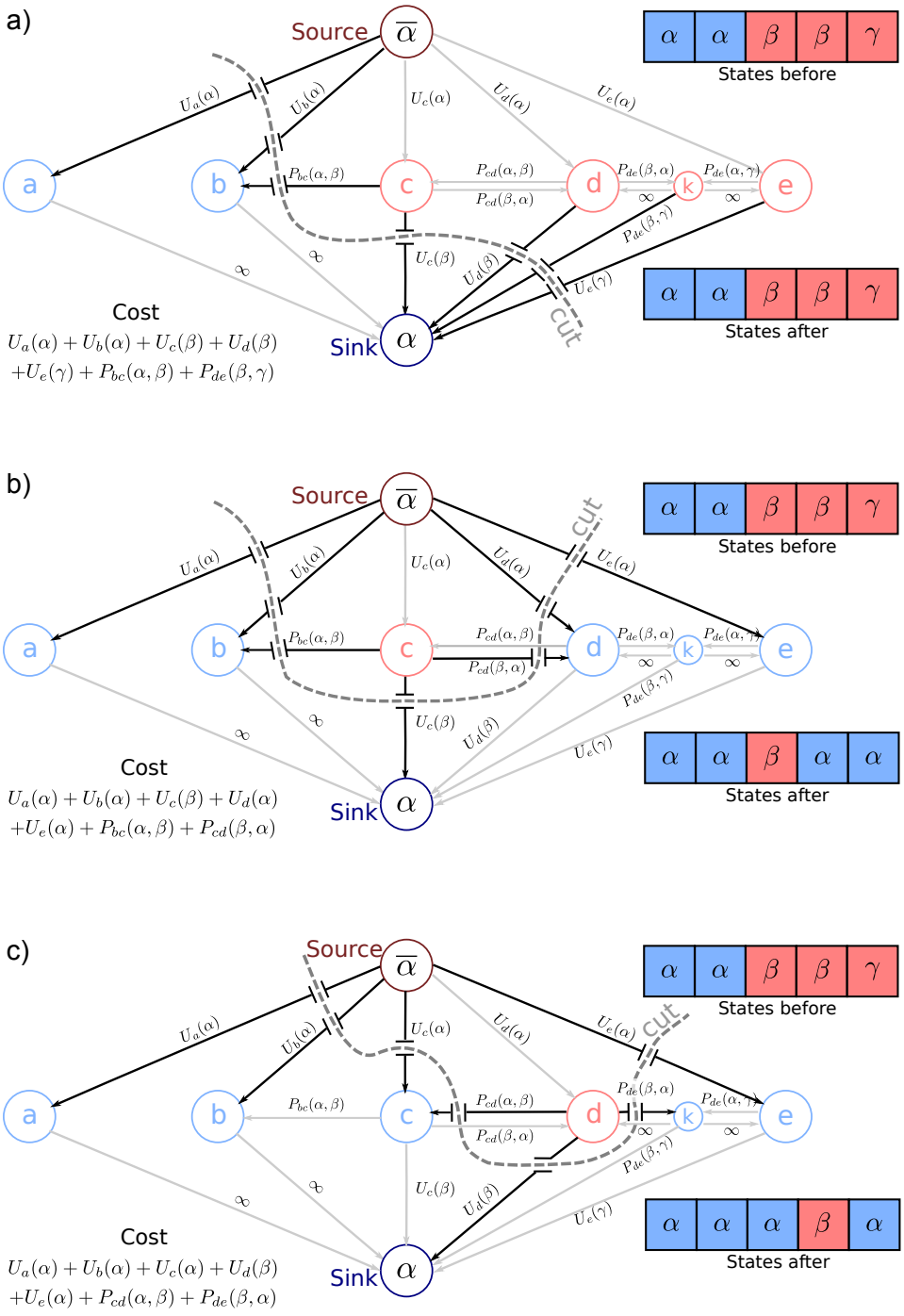

Figure 12.21 Alpha-expansion algorithm. a-c) Example cuts on this graph illustrate that the appropriate unary and pairwise costs are always paid.

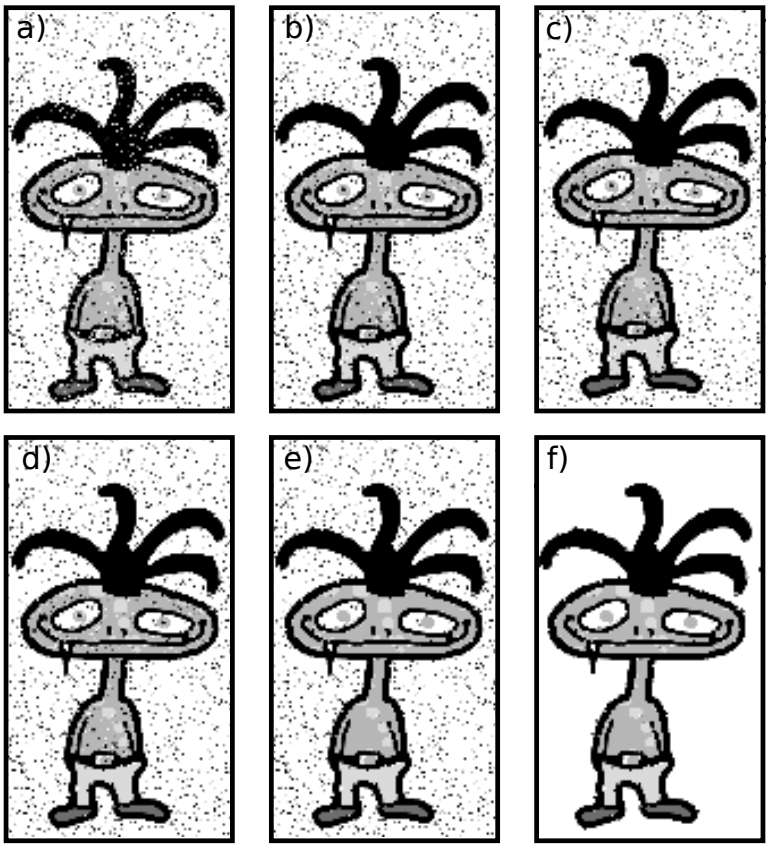

Figure 12.22 Alpha-expansion algorithm for denoising task. a) Observed noisy image. b) Label 1 (black) is expanded, removing noise from the hair. c-f) Subsequent iterations in which the labels corresponding to the boots, trousers, skin and background are expanded respectively.

encourage smoothness between neighboring labels, then the negative log posterior probability will again be the sum of unary and pairwise terms. The MAP labels $\hat{\mathbf{w}}$ can hence be found by minimizing a cost function of the form

$$\hat{\mathbf{w}} = \underset{w_{1\ldots N}}{\operatorname{argmin}} \left[\sum_{n=1}^{N} U_n(w_n) + \sum_{(m,n)\in\mathcal{C}} P_{mn}(w_m, w_n) \right], \tag{12.25}$$

and the graphical model will be as in figure 12.23. This cost function can be minimized using the graph cuts techniques described throughout this chapter.

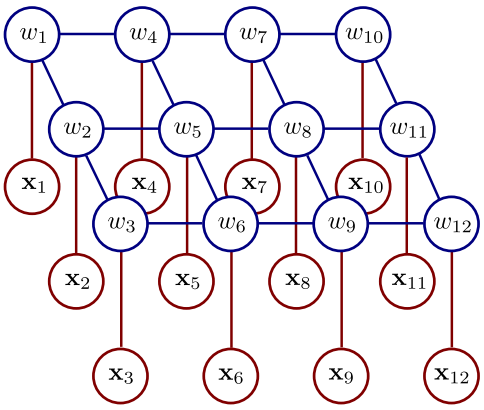

Figure 12.23 Graphical model for conditional random field (compare to figure 12.4). The posterior probability of the labels $\mathbf{w}$ is a Markov random field for fixed data $\mathbf{x}$. In this model, the two sets of cliques relate (i) neighbouring labels and (ii) each label to its associated measurement. Since this model only includes unary and pairwise interactions between the labels, the unknown labels $\{w_n\}_{n=1}^N$ can be optimized using graph cut techniques.

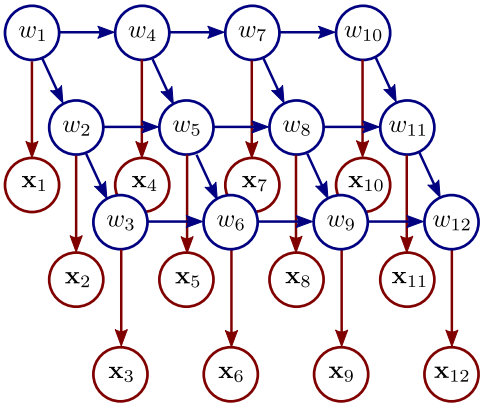

Figure 12.24 Directed graphical model for grid. Although this model appears similar to the pairwise Markov random field model, it represents a different factorization of the joint probability. In particular the factorization contains terms involving three variables such as $Pr(w_5|w_2, w_4)$. This means that the resulting cost function for MAP inference is no longer amenable to exact solution using graph-cut methods. In this case an attractive alternative is to use sampling based methods as it is easy to generate samples from this directed model.

12.6 Higher order models

The models that we have discussed so far have only connected immediate neighbors. However, these only allow us to model relatively simple statistical properties of the label field. One way to improve this situation is to consider each variable $w_n \in \{1 \ldots K\}$ as representing the index of a square patch of labels from a predefined library. The pairwise MRF now encodes the affinity of neighboring patches for each other. Unfortunately, the resulting costs are less likely to be submodular, or even obey the triangle inequality, and the number K of patches in the library is usually very large, making graph cut algorithms inefficient.

A second approach to modeling more complex statistical properties of the label field is to increase the number of the connections. For the undirected models (CRF, MRF) this would mean introducing larger cliques. For example, to model local texture, we might connect all of the variables in every 5×5 region of the image. Unfortunately, inference is hard in these models; optimizing the resulting complex cost functions is still an open research topic.

12.7 Directed models for grids

The Markov random field and conditional random field models are attractive because we can use graph-cuts approaches to search for the MAP solution. However, they have the drawback that it is very hard to learn the parameters of the model because they are based on undirected models. An obvious alternative is to use a similar directed model (figure 12.24). Here, learning is relatively easy, but it turns out that MAP inference using graph cuts is not generally possible.

To see this, consider the cost function for MAP inference in this model,

$$\begin{aligned}\hat{w}_{1\ldots N} &= \underset{w_{1\ldots N}}{\operatorname{argmax}} \left[\log[Pr(\mathbf{x}_{1\ldots N}|w_{1\ldots N})] + \log[Pr(w_{1\ldots N})]\right] \\ &= \underset{w_{1\ldots N}}{\operatorname{argmax}} \left[\sum_{n=1}^{N} \log[Pr(\mathbf{x}_n|w_n)] + \sum_{n=1}^{N} \log[Pr(w_n|w_{pa[n]})]\right] \\ &= \underset{w_{1\ldots N}}{\operatorname{argmin}} \left[\sum_{n=1}^{N} -\log[Pr(\mathbf{x}_n|w_n)] - \sum_{n=1}^{N} \log[Pr(w_n|w_{pa[n]})]\right],\end{aligned} \tag{12.26}$$

where we have multiplied the objective function by minus one and now seek the minimum. This minimization problem now has the general form

$$\hat{w}_{1\ldots N} = \underset{w_{1\ldots N}}{\operatorname{argmin}} \left[\sum_{n=1}^{N} U_n(w_n) + \sum_{n=1}^{N} T_n(w_n, w_{pa_1[n]}, w_{pa_2[n]})\right], \tag{12.27}$$

where $U_n(w_n)$ is called a *unary term* reflecting the fact that it only depends on a single element w_n of the label field and $T_n(w_n, w_{pa_1[n]}, w_{pa_2[n]})$ is called a *three-wise term* reflecting the fact in general the label at a pixel is conditioned on the two parents $pa_1[n]$ and $pa_2[n]$ above and to the left of the current position.

Notice that this cost function is fundamentally different from the cost function for MAP inference in a pairwise MRF (equation 12.11): it includes three-wise terms and there is no known polynomial algorithm to optimize this criterion. However, since this model is a directed graphical model, it is easy to generate samples from this model, and this can be exploited for approximate inference methods such as computing the empirical max marginals.

12.8 Applications

The models and algorithms in this chapter are used in a large number of computer vision applications, including stereo vision, motion estimation, background subtraction, interactive segmentation, semantic segmentation, image editing, image denoising, image super-resolution, and building 3D models. Here, we review a few key examples. We consider background subtraction which is a simple application with binary labels and interactive segmentation which uses binary labels in a system that simultaneously estimates the parameters in the likelihood terms. Then we

consider stereo, motion estimation, and image editing, all of which are multi-label graph-cut problems. We consider super-resolution which is a multi-label problem where the units are patches rather than pixels and which there are so many labels that the alpha-expansion algorithm is not suitable. Finally, we consider drawing samples from directed grid models to generate novel images.

12.8.1 Background subtraction

First, let us revisit the background subtraction algorithm that we first encountered in section 6.6.2. In background subtraction, the goal is to associate a binary label $\{w_n\}_{n=1}^N$ with each of the N pixels in the image, indicating whether this pixel belongs to the foreground or background, based on the observed RGB data $\{\mathbf{x}_n\}_{n=1}^N$ at each pixel. When the pixel is background ($w_n = 0$), the data are assumed to be generated from a normal distribution with known mean $\boldsymbol{\mu}_{n0}$ and covariance $\boldsymbol{\Sigma}_{n0}$. When the pixel is foreground ($w_n = 1$), a uniform distribution over the data is assumed, so that

$$\begin{aligned} Pr(\mathbf{x}_n|w=0) &= \text{Norm}_{\mathbf{x}_n}[\boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n] \\ Pr(\mathbf{x}_n|w=1) &= \kappa, \end{aligned} \tag{12.28}$$

where κ is a constant.

In the original description, we assumed that the models at each pixel were independent, and when we inferred the labels, the results were noisy (figure 12.25b). We now place a Markov random field prior over the binary labels where the pairwise cliques are organized as a grid (as in most of the models in this chapter) and where the potential functions encourage smoothness. Figure 12.25 illustrates the results of performing inference in this model using the graph cuts algorithm. There are now far fewer isolated foreground regions and fewer holes in the foreground object. The model has still erroneously discovered the shadow; a more sophisticated model would be required to deal with this problem.

12.8.2 Interactive segmentation (GrabCut)

The goal of interactive segmentation is to cut out the foreground object in a photo, based on some input from the user (figure 12.26). More precisely, we aim to associate a binary label $\{w_n\}_{n=1}^N$ to each of the N pixels in the image, indicating whether this pixel belongs to the foreground or background, based on the observed RGB data $\{\mathbf{x}_n\}_{n=1}^N$ at each pixel. However, unlike background subtraction, we do not have any prior knowledge of either the foreground or the background.

In the GrabCut system of Rother *et al.* (2005) the likelihoods of observing the background ($w = 0$) and foreground ($w = 1$) are each modeled as a mixture of K Gaussians so that

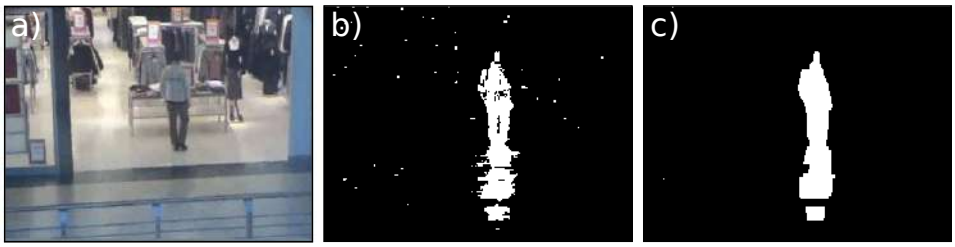

Figure 12.25 Background subtraction revisited. a) Original image. b) MAP solution of background subtraction model with independent pixels. The solution contains noise. c) MAP solution of background subtraction model with Markov random field prior. This smoothed solution has eliminated most of the noise.

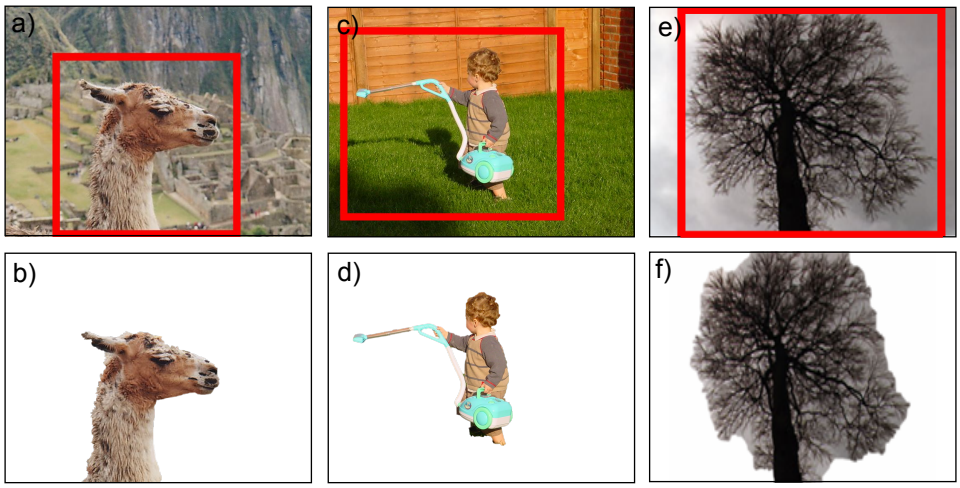

Figure 12.26 Grab Cut. a) The user draws a bounding box around the object of interest. b) The algorithm segments the foreground from the background by alternating between building color models and segmenting the image. c-d) A second example. e-f) Failure mode. This algorithm does not segment 'wiry' objects well as the pairwise costs for tracing around all the boundaries are prohibitive. Adapted from Rother *et al.* (2005). ©2005 ACM.

$$Pr(\mathbf{x}_n|w=j) = \sum_{k=1}^{K} \lambda_{jk} \text{Norm}_{\mathbf{x}_n}[\boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}], \tag{12.29}$$

and the prior over the labels is modeled as a pairwise connected Markov random field with the potentials chosen to encourage smoothness.

In this application, the image may have a wide variety of content, and so there is no suitable training data from which to learn the parameters $\{\boldsymbol{\lambda}_{jk}, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}\}_{j=1,k=1}^{2,K}$ of the foreground and background color models. However, we note that (i) if we knew the color models, we could perform the segmentation via MAP inference with the graph cuts algorithm and (ii) if we knew the segmentation, then we could compute the foreground and background color models based on the pixels assigned to each category. This observation leads to an alternating approach to inference in this model, in which the segmentation and parameters are computed in turn until the system converges.

In the *Grabcut* algorithm, the user draws a bounding box around the desired object to be segmented. This effectively defines a rough segmentation (pixels within the box are foreground, and pixels outside are background) from which the system is initialized. If the segmentation is not correct after the alternating optimization algorithm converges, the user may 'paint' regions of the image with a foreground or background brush, indicating that these *must* belong to the appropriate class in the final solution. In practice, this means that the unary costs are set to ensure that these take the appropriate values, and the alternating solution is run again from this point until convergence. Example results are shown in figure 12.26.

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' objects 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.

12.8.3 Stereo vision

In stereo vision, the goal is to infer a discrete multivalued label $\{w_n\}_{n=1}^N$ representing the disparity (horizontal shift) at each pixel in the image, given the observed image data $\{\mathbf{x}_n\}_{n=1}^N$. More details about the likelihood terms in this problem can be found in section 11.8.2, where we described tree-based priors for the unknown disparities. A more suitable approach is to use an MRF prior.

As for the denoising example, it is undesirable to use an MRF prior where the costs are a convex function of the difference in neighboring labels. This results in a MAP solution where the edges of objects are smoothed. Hence, it is usual to use a non-convex prior such as the Potts function, which embodies the idea that the scene consists of smooth surfaces, with sudden jumps in depth between them where the size of the jump is unimportant.

Boykov *et al.* (1999) used the alpha-expansion algorithm to perform approximate inference in a model of this sort (figure 12.27). The performance of this algorithm is good, but errors are found where there is no true match in the other image (i.e., where the corresponding point is occluded by another object). Kol-

Figure 12.27 Stereo vision. a) One image of the original stereo pair. b) Disparity estimated using the method of Boykov *et al.* (1999). c) Ground truth disparity. Blue pixels indicate regions which are occluded in the second image, and so do not have a valid match or disparity. The algorithm does not take account of this fact, and produces noisy estimates in these regions. Adapted from Boykov *et al.* (1999).

mogorov & Zabih (2001) subsequently developed a bespoke graph for dealing with occlusions in stereo vision, and an alpha-expansion algorithm for optimizing the associated cost function. These methods can also be applied to *optical flow* in which we attempt to identify pixel correspondences between adjacent frames in a video sequence. Unlike in stereo vision, there is no guarantee that the corresponding matches will be on the same scanline, but other than this, the problem is very similar.

12.8.4 Rearranging Images

Markov random field models can also be used 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 re-targeting*), remove an object or move an object from one place to another.

Pritch *et al.* (2009) constructed a model with variables $\mathbf{w} = \{w_1 \ldots w_N\}$ at each of the N pixels of $I^{(2)}$. Each possible value of $w_n \in \{1 \ldots K\}$ represents a 2D relative offset to image $I^{(1)}$ that tells us which pixel from image $I^{(1)}$ will appear at the n^{th} pixel of the new image. The label map $\mathbf{w}$ is hence termed a shift-map, as it represents 2D shifts to the original image. Each possible shift-map defines a different output image $I^{(2)}$ (figure 12.28).

Pritch *et al.* (2009) model the shift-map $\mathbf{w}$ as an MRF with pairwise costs that encourage smoothness. The result of this is that only shift-maps that are piecewise constant have high probability: in other words, new images which consist of large chunks of the original image that have been copied verbatim, are favored. They modify the pairwise costs, so that they are lower when adjacent labels 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 12.29):

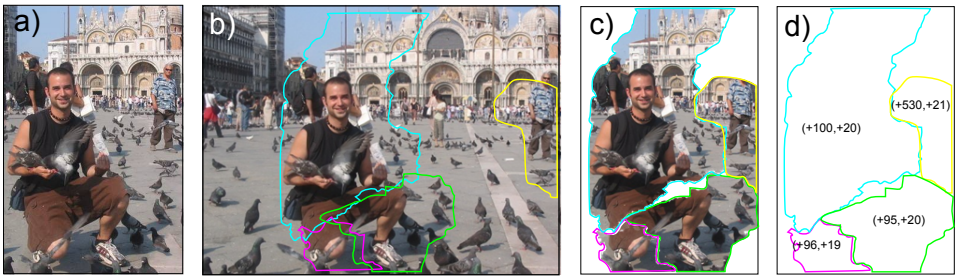

Figure 12.28 Shift maps for image re-targeting to reduce width. a) New image $I^{(2)}$ is created from b) the original image $I^{(1)}$ by copying piecewise regions (five regions shown). c) These regions are carefully chosen to produce a seamless result. d) The underlying representation is a shiftmap – 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. Figure shows method of Pritch *et al.* (2009).

- To *move* an object, we specify unary costs in the new region that ensure that we copy the desired object here. The remainder of the shifts are left free to vary, but favor 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 unary costs so that the remainder 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 the region.

- To *retarget* an 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* an image to a smaller width (figure 12.28), we additionally specify that the horizontal offset can only increase as we move from left to right. This ensures that the new image does not contain replicated objects and that their horizontal order remains constant.

In each case the best solution can be found using the alpha-expansion algorithm. Since the pairwise terms do not form a metric here, it is necessary to truncate the relevant costs (see section 12.4.1). In practice, there are many labels and so Pritch *et al.* (2009) introduce a *coarse-to-fine* scheme in which a low resolution version of the image is initially synthesized and the result of this is used to guide further refinements at higher resolutions.

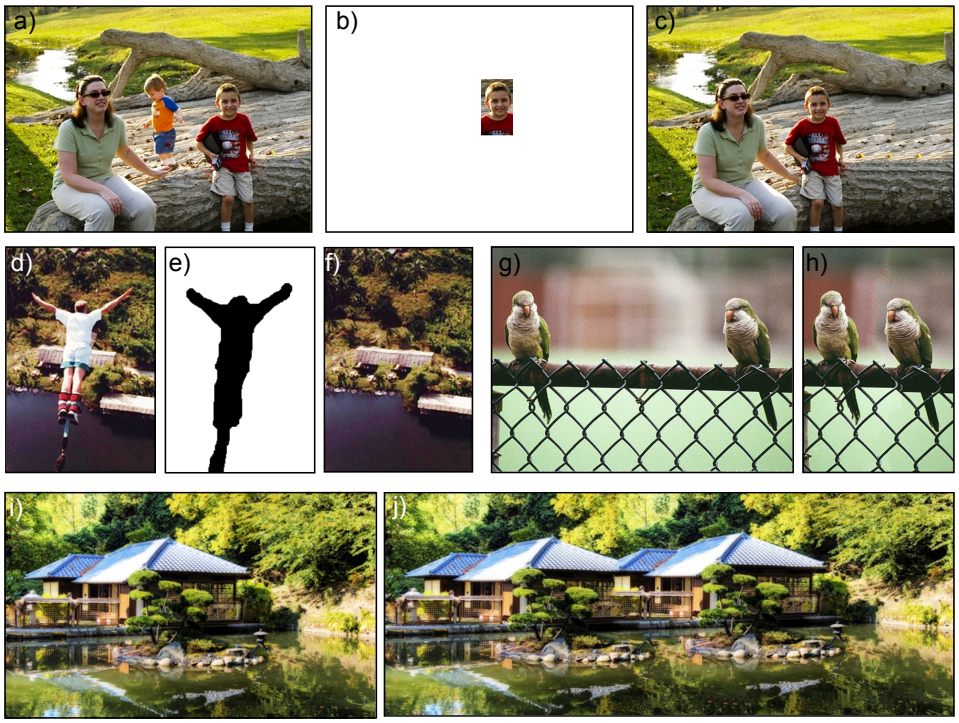

Figure 12.29 Applications of shift maps. Shift maps can be used to a) take an object from the original image b) move it to a new position and c) then fill in the remaining pixels to produce a new picture. d) They can also be used to remove an undesirable object e) specified by a mask from an image by f) filling in the missing area. g-h) Finally they can be used to retarget an original image to a smaller size, or i-j) to re-target an original image to a larger size. Results from method of Pritch *et al.* (2009).

12.8.5 Super-resolution

Image *super-resolution* can also be framed as inference within a Markov random field model. Here, the basic unit of currency is an image patch rather than a pixel. For example, consider dividing the original image into a regular grid of N low resolution 3×3 patches $\{\mathbf{x}_n\}_{n=1}^N$. The goal is to infer a set of corresponding labels $\{w_n\}_{n=1}^N$ at each position in the grid. Each label can take one of K values, each of which corresponds to a different possible high resolution 7×7 patch. These patches are extracted from training images.

The pairwise cost for placing high-resolution patches together is determined by the agreement at the abutting edge. The unary cost for choosing a patch at a given position depends on the agreement between the proposed high-resolution patch and the observed low-resolution patch. This can be computed by downsampling the high-resolution patch to 3×3 pixels, and then using a normal noise model.

Figure 12.30 Super resolution. a) The observed image, which is broken down into a regular grid of low resolution patches. b) We infer a regular grid of labels, each of which corresponds to a high resolution patch, and 'quilt' these together to form the super-resolved image. c) Ground truth. Adapted from Freeman *et al.* (2000). ©2000 Springer.

In principle, we could perform inference in this model with a graph cut formulation, but there are two problems. First, the resulting cost function is not submodular. Second, the number of possible high-resolution patches must be very large, and so the alpha-expansion algorithm (which chooses these in turn) would be extremely inefficient.

Freeman *et al.* (2000) used loopy belief propagation to perform approximate inference in a model similar to this. To make this relatively fast, they used only a subset of $J \ll K$ possible patches at each position, where these were chosen so that they were the J patches which agreed best with the observed data (and so had the lowest unary costs). Although the results (figure 12.30) are quite convincing, they are sadly far from the imagined feats of TV crime drama.

12.8.6 Texture synthesis

The applications so far have all been based on performing inference in the undirected Markov random field model. We now consider the directed model. Inference is difficult in this model due to the presence of 'three-wise' terms in the associated cost function (see section 12.7). However, generation from this model is relatively easy; since this is a directed model, we can use an ancestral sampling technique to generate examples. One possible application of this technique is for *texture synthesis*.

The goal of texture synthesis is to learn a generative model from a small patch of texture such that when we draw samples from the model, they look like extended examples of the same texture (figure 12.31). The particular technique that we describe here is known as *image quilting* and was originally described by Efros & Freeman (2001). We will first describe the algorithm as it was initially conceived, and then relate it to the directed model for grids.

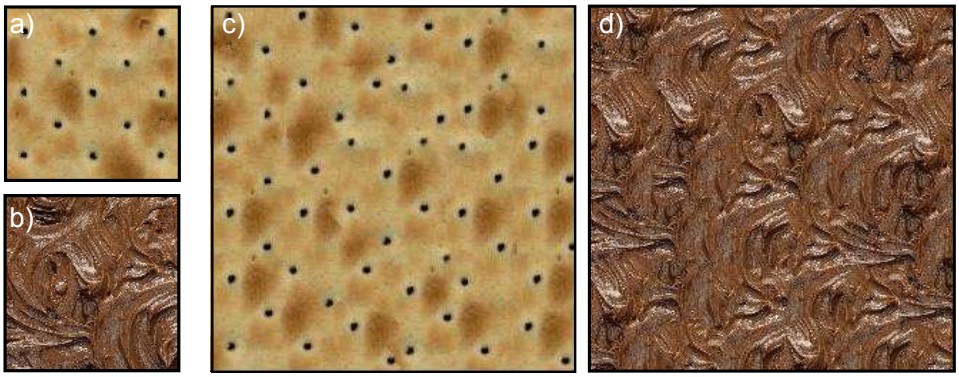

Figure 12.31 Texture synthesis. a,b) Original texture samples. c,d) Synthesized textures using *image quilting*. Adapted from Efros & Freeman (2001).

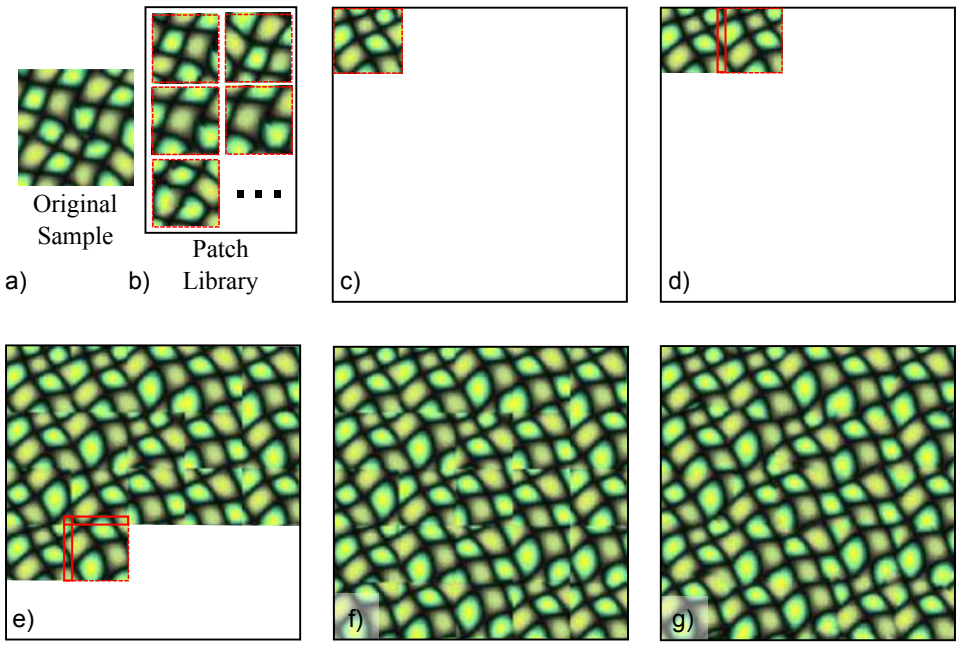

Figure 12.32 Image quilting. a) Original texture sample. b) Library of all overlapping patches from the original texture sample. c) The first patch is chosen randomly from the library. d) The second patch is chosen randomly from the k library patches that are most similar in the overlapping region. e) In subsequent rows, patches are chosen so that the overlapping region agrees with the previously placed patches to the left and above. f) This continues until we reach the bottom-right of the image. g) The patches are then blended together to give the final results.

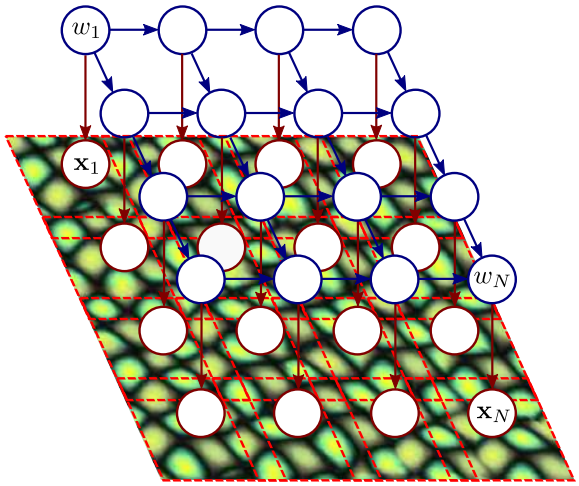

Figure 12.33 Image quilting as ancestral sampling from a graphical model. When we synthesize images, we are effectively ancestral sampling from a directed grid model where each hidden node represents a patch index and each observed variable represents the patch data.

The first step (see figure 12.32) is to extract all possible patches of a given size from the input texture to form a *patch library*. The synthesized image will consist of a regular grid of these library patches such that each overlaps its neighbors by a few pixels. A new texture is synthesized starting in the top-left of this grid and proceeding to the bottom-right. At each position, a library patch is chosen such that it is visually consistent with the patches that have previously been placed above and to the left.

For the top-left position, we randomly choose a patch from the library. We then consider placing a second patch to the right of the first patch, such that they overlap by roughly 1/6 of their width. We search through the library for the J patches where the squared RGB intensity difference in the overlapping region is smallest. We choose one of these J patches randomly and place it into the image at the second position. We continue in this way, synthesizing the top row of patches in the image. When we reach the second row, we must consider the overlap with the patches to the left and above in deciding whether a candidate library patch is suitable: we choose the J patches where the total RGB difference between the overlapping portions of the candidate patch and the previously chosen patches is minimal. This process continues until we reach the bottom-right of the image.

In this way, we synthesize a new example of the texture (figure 12.32a-f). By forcing the overlapping regions to be similar, we enforce visual consistency between adjacent patches. By choosing randomly from the J best patches, we ensure that the result is stochastic: if we always chose the most visually consistent patch, we would replicate the original texture verbatim. At the end of this process, it is common to blend the resulting patches together to remove remaining artifacts in

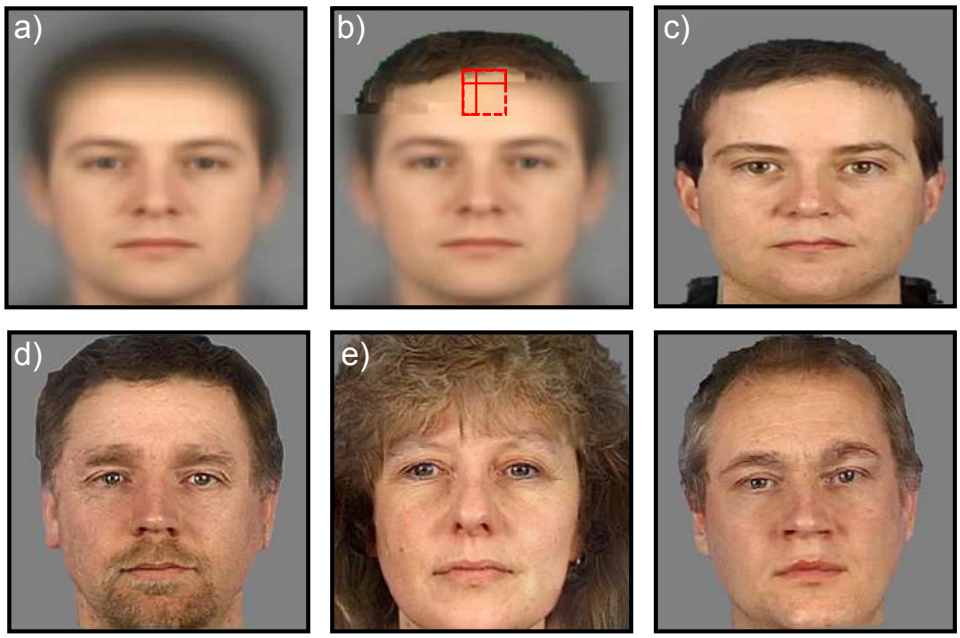

Figure 12.34 Synthesizing novel faces. a) A sample is drawn from a subspace model (see chapter 7) that has been trained on facial images. b) Texture synthesis now proceeds but with two differences from before. First, the choice of patch must now agree with the sample from the subspace model as well as the previously placed patches. Second, the library patches are now different at each position: in this way we ensure that a nose patch is always chosen in the center and so on. c) After completing the synthesis and blending together the patches. d-f) Three more examples of synthesized faces. Adapted from Mohammed *et al.* (2009). ©2009 ACM.

the overlapping region (figure 12.32g).

Image quilting can be thought of as ancestral sampling from the directed model for images (figure 12.33). The observed data $\{\mathbf{x}_n\}_{n=1}^N$ are the output patches, and the hidden labels $\{w_n\}_{n=1}^N$ represent the patch index. The labels are conditioned on their parents with a probability distribution that allots a constant probability if the overlapping region is one of the J closest and zero otherwise. The only real change is that the relationship between label and observed data is now deterministic: a given label always produces exactly the same output patch.

12.8.7 Synthesizing novel faces

Mohammed *et al.* (2009) presented a related technique to synthesize more complex objects, such as frontal faces (figure 12.35), based on a large database of weakly aligned training examples. Faces have a distinct spatial structure, and we must

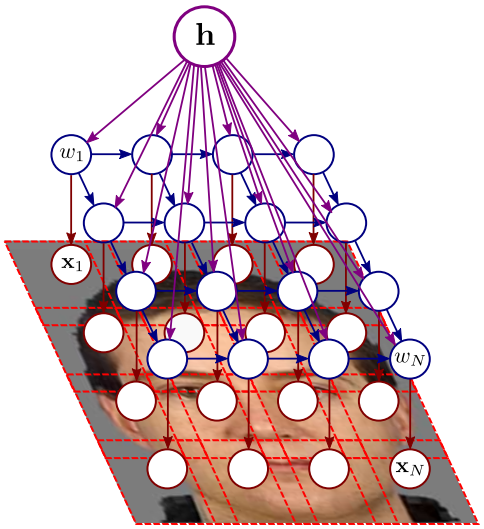

Figure 12.35 Graphical model for synthesizing novel faces. When we generate a new image we are ancestral sampling from a directed image model, where each label w is conditioned on the hidden variable $\mathbf{h}$ of the subspace model. Adapted from Mohammed *et al.* (2009). ©2009 ACM.

ensure that our model enforces these constraints. To this end, we build a separate library of patches for each position in the image. This ensures that the features have roughly the correct spatial relations: the nose always appears in the center and the chin at the bottom.

In principle, we could now apply a standard image quilting approach by synthesizing patches starting in the top-left, and moving to the bottom-right. Unfortunately, the resulting faces can drift in appearance (e.g., from male to female) as we move through the image. To prevent this from happening, we condition the patch synthesis on a draw from a factor analysis model (section 7.6) which has been trained with frontal faces. A sample from this model looks like a blurry, but globally coherent, face. Now when we choose potential patches, they must agree with both the previously placed patches to the left and above, but also be similar to the appropriate part of the blurry sample from the subspace model. The generated images from this model look like highly realistic human faces.

In terms of probability, the labels $\{w_n\}_{n=1}^N$ in this model are conditioned not only on their ancestors w_{pa}, but also on the hidden variable in the subspace model $\mathbf{h}$. This connects to every patch label $\{w_n\}_{n=1}^N$ and gives the resulting image a greater visual coherence than the Markov connections of the patches alone.

Discussion

Models for grids are ubiquitous in vision: they occur in almost all applications that attempt to associate a label with each position in the image. Depending on the application, this label may indicate the depth, object type, segmentation mask, or motion at that pixel. Unfortunately, most problems of this type are NP hard and so we must resort to efficient approximate inference techniques such as the alpha-expansion algorithm.

Notes

MRFs and CRFs: Markov random fields were first investigated in computer vision by Geman & Geman (1984), although much of the early work dealt with continuous variables rather than the discrete case as discussed in this chapter. A good review can be found in Li (2010). Conditional random fields were first used in computer vision by Kumar & Hebert (2003). An overview can be found in Sutton & McCallum (2011).

Applications: Grid-based models and graph cuts are used extensively in vision and graphics. A partial list of applications includes stereo vision (Kolmogorov & Zabih 2001; Woodford *et al.* 2009), optical flow (Kolmogorov & Zabih 2001), texture synthesis (Kwatra *et al.* 2003), photo-montage (Agarwala *et al.* 2004), summarizing photo-collections with collages (Rother *et al.* 2005; Rother *et al.* 2006), bi-layer segmentation (Kolmogorov *et al.* 2006), interactive segmentation (Rother *et al.* 2004; Boykov *et al.* 2001), super-resolution (Freeman *et al.* 2000), image re-targeting (Pritch *et al.* 2009), denoising (Greig *et al.* 1989), over-segmentation (Moore *et al.* 2010; Veksler *et al.* 2010), image colorization (Levin *et al.* 2004), segmantic segmentation (Shotton *et al.* 2009), multi-view reconstruction (Kolmogorov & Zabih 2002; Vogiatzis *et al.* 2007), and matching image points (Isack & Boykov 2012).

Graph cuts: The first application of graph cuts to inference in an MRF is due to Greig *et al.* (1989) who investigated binary denoising. However, it was not until the work of Boykov *et al.* (2001) that this result was rediscovered and graph cuts became widely used. Ishikawa (2003) presented the exact solution for multi-label graph cuts with convex potentials, and this was generalized by Schlesinger & Flach (2006). The presentation in this chapter is a hybrid of these two methods. Boykov *et al.* (2001) introduced the idea of optimizing non-convex multi-label energies via a series of binary problems. They proposed two algorithms of this kind: the alpha-beta swap in which pairs of labels are exchanged for one another and the alpha-expansion algorithm. They also proved that the alpha-expansion solution is guaranteed to be within a factor of two of the true solution. In the same spirit, Lempitsky *et al.* (2010) and Kumar *et al.* (2011) have proposed more complex 'moves'. Tarlow *et al.* (2011) elucidates the conection between graph cut methods and max-product belief propagation. For more detailed overviews of graph cut methods, consult Boykov & Veksler (2006), Felzenszwalb & Zabih (2011) and Blake *et al.* (2011).

Max-flow: Graph-cut methods rely on algorithms for computing maximum flow. The most common of these are the augmenting paths method of Ford & Fulkerson (1962) and the push-relabel method of Goldberg & Tarjan (1988). Details of these and other approaches to the same problem can be found in any standard textbook on algorithms such as Cormen *et al.* (2001). The most common technique in computer vision is a modified version of the augmented paths algorithm due to Boykov & Kolmogorov (2004) that has been demonstrated to have very good performance for vision problems. Kohli & Torr (2005), Juan & Boykov (2006) and Alahari *et al.* (2008) have all investigated methods for improving the efficiency of graph cuts by reusing solutions to similar graph cut problems (e.g., based on the solution to the previous frames in a time-sequence).

Cost functions and optimization: Kolmogorov & Zabih (2004) provide a summary of the cost functions that can be optimized using the basic graph cuts max flow formulation with binary variables. Kolmogorov & Rother (2007) summarize graph cut approaches to non-submodular energies. Rother *et al.* (2007) and Komodakis *et al.* (2008) present algorithms that can approximately optimize more general cost functions.

Constraint edges: Recent work has investigated bespoke graph constructions that make heavy use of constraint edges (edges of infinite strength) to ensure that the solution

conforms to a certain structure. For example, Delong & Boykov (2009) devised a method that forced certain labels to surround others, and Moore *et al.* (2010) describe a method that forces the label field to conform to a lattice. See also Felzenszwalb & Veksler (2010) for a related scheme based on dynamic programming.

Higher-order cliques: All of the methods discussed in this chapter assume pairwise connections; the cliques include only two discrete variables. However, to model more complex statistics of the label field, it is necessary to include more than two variables in the cliques, and these are known as *higher order* models. Roth & Black (2009) demonstrated good denoising and inpainting results with a continuous MRF model of this kind, and Domke *et al.* (2008) demonstrated the efficacy of a directed model in which each variable was conditioned on a number of variables above and to the right in the image. There has recently been considerable interest in developing algorithms for MAP estimation in models with discrete variables and higher-order cliques (Ishikawa 2009; Kohli *et al.* 2009a; Kohli *et al.* 2009b; Rother *et al.* 2009).

Other approaches to MAP estimation: There are many other contemporary approaches to MAP estimation in MRFs and CRFs. These include loopy belief propagation (Weiss & Freeman 2001), quadratic pseudo-boolean optimization which is used in non-submodular cost functions (Kolmogorov & Rother 2007), random walks (Grady 2006), and linear programming (LP) relaxations (Weiss *et al.* 2011) and various approaches to maximize the LP lower bound such as tree reweighted message passing (Wainright *et al.* 2005; Kolmogorov 2006). An experimental comparison between different energy minimization methods for MRFs can be found in Szeliski *et al.* (2008).

Texture synthesis: Texture synthesis was originally investigated as a continuous problem and the focus was on modeling the joint statistics of the RGB values in a small patch (Heeger & Bergen 1995; Portilla & Simoncelli 2000). Although texture synthesis as a continuous problem is still an active research area (e.g., Heess *et al.* 2009), these early methods were displaced by methods that represented the texture in terms of discrete variables (either by quantizing the RGB values, indexing patches or using a shift-map representation). The resulting algorithms (e.g., Efros & Leung 1999; Wei & Levoy 2000; Efros & Freeman 2001; Kwatra *et al.* 2003) were originally described as heuristic approaches to generating textures, but can also be interpreted as exact or approximate ways to draw samples from directed or undirected grid models.

Interactive segmentation: The use of graph cuts for interactive segmentation algorithms was pioneered by Boykov & Jolly (2001). In early works (Boykov & Jolly 2001; Boykov & Funka Lea 2006; Li *et al.* 2004) the user interacted with the image by placing marks indicating foreground and background regions. Grab-cut (Rother *et al.* 2004) allowed the user to draw a box around the object in question. More recent systems (Liu *et al.* 2009) are fast enough to allow the user to interactively 'paint' the selection onto the images. Current interest in graph cut based segmentation is mainly focused on developing novel priors over the shape that improve performance (e.g., Malcolm *et al.* 2007; Veksler 2008; Chittajallu *et al.* 2010; Freiman *et al.* 2010). To this end, Kumar *et al.* (2005) introduced a method for imposing high-level knowledge about the articulation of the object, Vicente *et al.* (2008) developed an algorithm that is suited for cutting out elongated objects, and Lempitsky *et al.* (2008) used a prior based on a bounding box around the object.

Stereo vision: Most state-of-the-art stereo vision algorithms rely on MRFs or CRFs and are solved using either graph cuts (e.g., Kolmogorov & Zabih 2001) or belief propagation (e.g., Sun *et al.* 2003). Comparisons of these approaches can be found in Tappen & Freeman (2003) and Szeliski *et al.* (2008). An active area of research in dense stereo vision is the formulation of the compatibility of the two images given a certain disparity

offset (e.g., Bleyer & Chambon 2010; Hirschmüller & Scharstein 2009) which is rarely based on single pixels in practice (see Yoon & Kweon 2006; Tombari *et al.* 2008).

For more information about stereo vision see the reviews by Scharstein & Szeliski (2002) and Brown *et al.* (2003) or consult Szeliski (2010), which contains a good modern summary. Chapter 11 of this book summarizes dynamic programming approaches. Notable stereo implementations include the region growing approach of Lhuillier & Quan (2002); the systems of Zitnick & Kanade (2000) and Hirschmüller (2005), both of which are available online; and the extremely efficient GPU based system of Sizintsev & Wildes (2010). For an up-to-date quantitative comparison of the latest stereo vision algorithms consult the Middlebury stereo vision website (http://vision.middlebury.edu/stereo/).

Problems

Problem 12.1 Consider a Markov random field with the structure

$$Pr(x_1, x_2, x_3, x_4) = \frac{1}{Z}\phi[x_1, x_2]\phi[x_2, x_3]\phi[x_3, x_4]\phi[x_4, x_1]$$

but where the variables x_1, x_2, x_3, and x_4 are continuous and the potentials are defined as

$$\phi[a, b] = \exp\left[-(a-b)^2\right].$$

This is known as a *Gaussian Markov random field*. Show that the joint probability is a normal distribution, and find the information matrix (inverse covariance matrix).

Problem 12.2 Compute the MAP solution to the three-pixel graph cut problem in figure 12.36 by (i) computing the cost of all eight possible solutions explicitly and finding the one with the minimum cost (ii) running the augmenting paths algorithm on this graph by hand and interpreting the minimum cut.

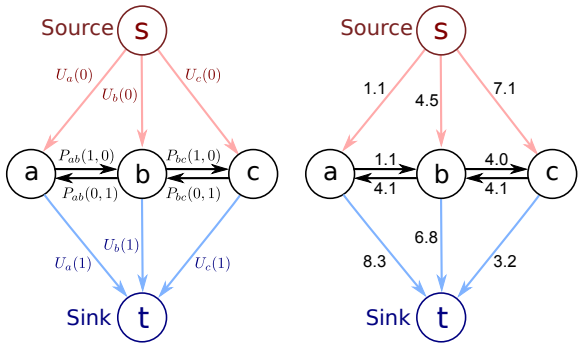

Figure 12.36 Graph for problem 12.2.

Problem 12.3 Explicitly compute the costs associated with the four possible minimum cuts of the graph in figure 12.10.

Problem 12.4 Compute the cost for each the four possible cuts of the graph in figure 12.11c.

Problem 12.5 Consider the graph construction in figure 12.37a, which contains a number of constraint edges of infinite cost (capacity). There are 25 possible minimum cuts on this graph, each of which corresponds to one possible labeling of the two pixels. Write out the cost for each labeling. Which solutions have finite cost for this graph construction?

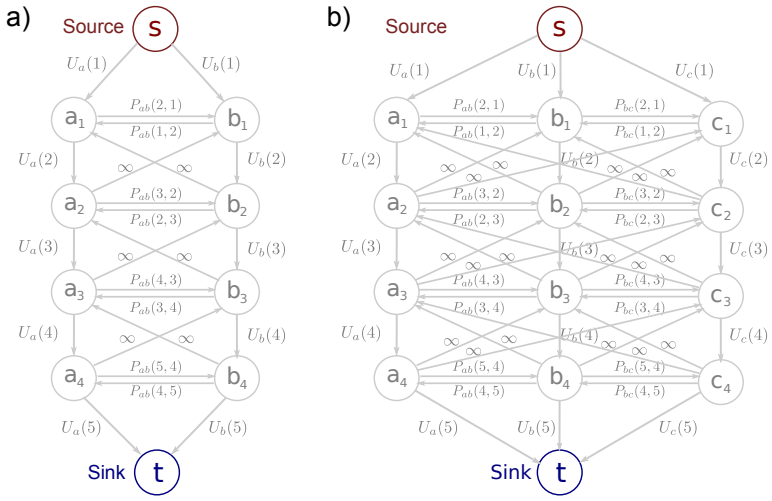

Figure 12.37 Alternative multi-label graph constructions. Each of these these graphs has extra *constraint links* with infinite weight. These have the effect of giving an infinite cost to a subset of the possible solutions.

Problem 12.6 Which of the possible minimum cuts of the graph in figure 12.37b have a finite cost?

Problem 12.7 Confirm that the costs of the cuts in figure 12.14 are as claimed by explicitly performing the summation over the relevant terms C_{ij}.

Problem 12.8 Show that the Potts model (figure 12.17c) is not submodular by providing a counter-example to the required criterion:

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

Problem 12.9 An alternative to the alpha-expansion algorithm is the alpha-beta swap. Here, a multi-label MRF with non-convex potentials is optimized by repeatedly choosing pairs of labels α, β and performing a binary graph cut that allows them to swap in such a way that the overall cost function decreases. Devise a graph structure that can be used to perform this operation. Hint: consider separate cases for neighboring labels (α,α), (β,β), (β,γ), (α,γ) and (γ,γ) where γ is a label that is neither α nor β.

Part IV

Preprocessing

Part IV: Preprocessing

The main focus of this book is on statistical models for computer vision; the previous chapters concern models that relate visual measurements $\mathbf{x}$ to the world $\mathbf{w}$. However, there has been little discussion of how the measurement vector $\mathbf{x}$ was created, and it has often been implied that it contains concatenated RGB pixel values. In state-of-the-art vision systems, the image pixel data are almost always *preprocessed* to form the measurement vector.

We define preprocessing to be any transformation of the pixel data *prior to* building the model that relates the data to the world. Such transformations are often ad-hoc heuristics: their parameters are not learned from training data, but they are chosen based on experience of what works well. The philosophy behind image preprocessing is easy to understand; the image data may be contingent on many aspects of the real world that do not pertain to the task at hand. For example, in an object detection task, the RGB values will change depending on the camera gain, illumination, object pose and particular instance of the object. The goal of image preprocessing is to remove as much of this unwanted variation as possible, while retaining the aspects of the image that are critical to the final decision.

In a sense, the need for preprocessing represents a failure; we are admitting that we cannot directly model the relationship between the RGB values and the world state. Inevitably, we must pay a price for this. Although the variation due to extraneous factors is jettisoned, it is very probable that some of the task-related information is also discarded. Fortunately, in these nascent years of computer vision, this rarely seems to be the limiting factor that governs the overall performance.

We devote the single chapter in this section to discussing a variety of preprocessing techniques. Although the treatment here is not extensive, it should be emphasized that preprocessing is very important; in practice, the choice of preprocessing technique can influence the performance of vision systems at least as much as the choice of model.