Foundations and Trends® in
Computer Graphics and Vision
Vol. 6, Nos. 3–4 (2010) 185–365

DOI: 10.1561/0600000033

Structured Learning and Prediction in Computer Vision

By Sebastian Nowozin and Christoph H. Lampert

Contents

Foundations and Trends® in
Computer Graphics and Vision
Vol. 6, Nos. 3–4 (2010) 185–365

DOI: 10.1561/0600000033

Structured Learning and Prediction in Computer Vision

Sebastian Nowozin[1] and Christoph H. Lampert [2]

[1] *Microsoft Research Cambridge, United Kingdom,*
Sebastian.Nowozin@microsoft.com
[2] *IST Austria, Institute of Science and Technology Austria, Austria,*
chl@ist.ac.at

Abstract

Powerful statistical models that can be learned efficiently from large amounts of data are currently revolutionizing computer vision. These models possess a rich internal structure reflecting task-specific relations and constraints. This monograph introduces the reader to the most popular classes of structured models in computer vision. Our focus is discrete undirected graphical models which we cover in detail together with a description of algorithms for both probabilistic inference and maximum a posteriori inference. We discuss separately recently successful techniques for prediction in general structured models. In the second part of this monograph we describe methods for parameter learning where we distinguish the classic maximum likelihood based methods from the more recent prediction-based parameter learning methods. We highlight developments to enhance current models and discuss kernelized models and latent variable models. To make the monograph more practical and to provide links to further study we provide examples of successful application of many methods in the computer vision literature.

1

Introduction

In a very general sense *computer vision* is about automated systems making sense of image data by extracting some high-level information from it. The *image data* can come in a large variety of formats and modalities. It can be a single natural image, or it can be a multi-spectral satellite image series recorded over time. Likewise, the *high-level information* to be recovered is diverse, ranging from physical properties such as the surface normal at each image pixel to object-level attributes such as its general object class ("car," "pedestrian," etc.).

The above task is achieved by building a *model* relating the image data to the high-level information. The model is represented by a set of variables that can be divided into the *observation variables* describing the image data, the *output variables* defining the high-level information, and optionally a set of additional *auxiliary variables*. Besides the variables a model defines how the variables *interact* with each other. Together the variables and interactions form the *structure* of the model.

Structured models allow a large number of variables and interactions, leading to rich models that are able to represent the complex relationships that exist between the image data and the quantities of interest.

Instead of specifying a single-fixed model we can also introduce free *parameters* into the interactions. Given some annotated data with known values for the output variables we can then adjust the parameters to effectively *learn* a good mapping between observation and output variables. This is known as *parameter learning* and *training the model*.

1.1 An Example: Image Segmentation

We will now use the task of foreground–background image segmentation to make concrete the abstract concepts just discussed. In foreground–background image segmentation we are given a natural image and need to determine for each pixel whether it represents the foreground object or the background. To this end we define one binary output variable $y_i \in \{0,1\}$ for each pixel i, taking $y_i = 1$ if i belongs to the foreground, $y_i = 0$ otherwise. A single observation variable $x \in \mathcal{X}$ will represent the entire observed image.

To define the interactions between the variables we consider the following: if the image around a pixel i looks like a part of the foreground object, then $y_i = 1$ should be preferred over $y_i = 0$. More generally we may assume a local model $g_i(y_i,x)$, where $g_i(1,x)$ takes a high value if x looks like a foreground object around pixel i, and a low value otherwise. If this were the only component of the model we would make independent decisions for each pixel. But this is clearly insufficient. For example the model g_i might be inaccurate or the image locally really does resemble the foreground object. Therefore we introduce an interaction aimed at making locally consistent decisions about the output variables: for each pair (i,j) of pixels that are close to each other in the image plane — say within the 4-neighborhood $\mathcal{J}$ — we introduce a pairwise interaction term $g_{i,j}(y_i,y_j)$ that takes a large value if $y_i = y_j$ and a small value otherwise.

We can now pose segmentation as a maximization problem over all possible segmentations on n pixels,

$$y^* = \operatorname*{argmax}_{y \in \{0,1\}^n} \left[\sum_{i=1}^{n} g_i(y_i,x) + \sum_{(i,j) \in \mathcal{J}} g_{i,j}(y_i,y_j) \right]. \qquad (1.1)$$

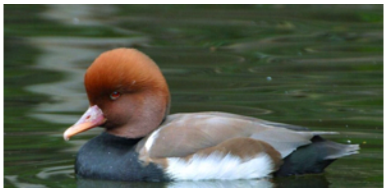


Fig. 1.1 Input image to be segmented into foreground and background. (Image source: `http://pdphoto.org`).

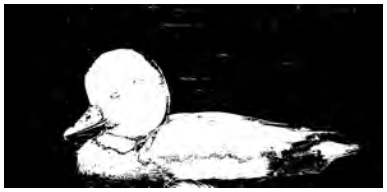


Fig. 1.2 Pixelwise separate classification by g_i only: noisy, locally inconsistent decisions.

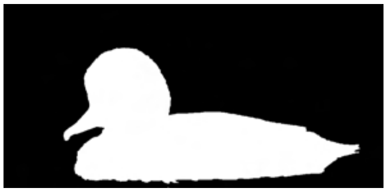


Fig. 1.3 Joint optimum y^* with spatially consistent decisions.

The optimal prediction y^* will trade off the quality of the local model g_i with making decisions that are spatially consistent according to $g_{i,j}$. This is shown in Figures 1.1 to 1.3.

We did not say how the functions g_i and $g_{i,j}$ can be defined. In the above model we would use a simple binary classification model

$$g_i(y_i, x) = \langle w_{y_i}, \varphi_i(x) \rangle, \tag{1.2}$$

where $\varphi_i : \mathcal{X} \to \mathbb{R}^d$ extracts some image features from the image around pixel i, for example color or gradient histograms in a fixed window around i. The parameter vector $w_y \in \mathbb{R}^d$ weights these features. This allows the local model to represent interactions such as "if the picture around i is green, then it is more likely to be a background pixel." By adjusting $w = (w_0, w_1)$ suitably, a local score $g_i(y_i, x)$ can be computed for any given image. For the pairwise interaction $g_{i,j}(y_i, y_j)$ we ignore the image x and use a 2×2 table of values for $g_{i,j}(0,0)$, $g_{i,j}(0,1)$, $g_{i,j}(1,0)$, and $g_{i,j}(1,1)$, for all adjacent pixels $(i,j) \in \mathcal{J}$.

1.2 Outline

In *Graphical Models* we introduce an important class of discrete structured models that can be concisely represented in terms of a graph. In this and later parts we will use *factor graphs*, a useful special class of graphical models. We do not address in detail the important class of *directed* graphical models and temporal models.

Computation in undirected discrete factor graphs in terms of probabilities is described in *Inference in Graphical Models.* Because for most models exact computations are intractable, we discuss a number of popular approximations such as belief propagation, mean field, and Monte Carlo approaches.

In *Structured Prediction* we generalize prediction with graphical models to the general case where a prediction is made by maximizing an arbitrary evaluation function, i.e., $y = f(x) = \mathrm{argmax}_y\, g(x,y)$. Solving this problem — that is, evaluating $f(x)$ — is often intractable as well and we discuss general methods to approximately make predictions.

After having addressed these basic inference problems we consider learning of structured models. In *Conditional Random Fields* we introduce popular learning methods for graphical models. In particular we focus on recently proposed efficient methods able to scale to large training sets.

In *Structured Support Vector Machines* we show that learning is also possible in the general case where the model does not represent a probability distribution. We describe the most popular techniques and discuss in detail the structured support vector machine.

Throughout the monograph we interleave the main text with successful computer vision applications of the explained techniques. For convenience the reader can find a summary of the notation used at the end of the monograph.

2

Graphical Models

In computer vision we often need to build a model of the real world that relates observed measurements to quantities of interest. For example, given an observed image we would like to know physical quantities like the depth from the observer for each measured pixel in the image. Alternatively we could be interested high-level questions such as knowing where all objects of a certain kind are visible in the image.

Graphical models allow us to encode relationships between multiple variables using a concise, well-defined language. We can use this language to relate observations and unknown variables to each other. Naturally some of the problems we are interested in solving are *not well-posed* in the sense that it is impossible to determine with certainty the correct answer from the observation. *Probabilistic graphical models* help in this setting. They encode a joint or conditional probability distribution such that given some observations we are provided not just with a single estimate of the solution but with a full probability distribution over all feasible solutions. Moreover, we can incorporate additional assumptions in the form of *prior probability distributions*.

There exist different types of graphical models, but they all have in common that they specify a family of probability distributions by means of a graph. The various types differ by the allowed graph structure and the *conditional independence assumptions* encoded in the graph. Given a graphical model we can think of it as a *filter for probability distributions* through which only distributions may pass that satisfy all the conditional independences encoded in the graph. Therefore a graphical model does not specify a single distribution but a family of probability distributions.

In this monograph we will limit ourselves to graphical models for discrete variables. One reason is that discrete models have been more popular in computer vision. Another reason is that in many cases prior assumptions and constraints on predictions are more easily enforced on discrete variables than on continuous ones. That said, models for continuous random variables are important because discretization of a continuous variable can lead to inefficiencies both in the computational and statistical sense. Computationally a fine discretization is needed to achieve high accuracy leading to a large and inefficient state space of the resulting model. Statistically, estimating many parameters for distinct states discards the original continuous structure that produced the discretization and leads to an increase in estimation error. Despite these conceptual drawbacks we feel that many problems in computer vision are best modeled using a discrete model.

We denote the set of output variables with V and the overall *output domain* by $\mathcal{Y}$. The output domain is the product of individual variable domains $\mathcal{Y}_i$ so that we have $\mathcal{Y} = \bigtimes_{i \in V} \mathcal{Y}_i$. In many practical models we have $\mathcal{Y}_i := \mathcal{L}$, a single set $\mathcal{L}$ of labels. The *input domain* $\mathcal{X}$ varies for different tasks; typically $\mathcal{X}$ would be the set of images and a single $x \in \mathcal{X}$ is one image. The random variables of the model are denoted by Y_i, and their realization is denoted by $Y_i = y_i$ or just y_i. Similarly, $Y = y$ or just y is the joint realization over all variables.

Two types of graphical models are popular. *Directed graphical models*, also known as Bayesian networks, specify the family $p(Y = y)$ $= p(y)$ by means of a directed acyclic graph $G = (V, \mathcal{E})$ and the factorization of $p(y)$ as

$$p(y) = \prod_{i \in V} p(y_i | y_{\mathrm{pa}_G(i)}), \tag{2.1}$$

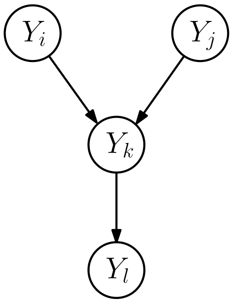


Fig. 2.1 Illustration of a Bayesian network. The directed acyclic graph defines the factorization as $p(Y = y) = p(Y_l = y_l|Y_k = y_k)p(Y_k = y_k|Y_i = y_i, Y_j = y_j)p(Y_i = y_i)p(Y_j = y_j)$.

where each $p(y_i|y_{\mathrm{pa}_G(i)})$ is a *conditional probability distribution*, and $\mathrm{pa}_G(i)$ denotes the set of parents of node $i \in V$. Figure 2.1 shows a simple Bayesian network defining a family of distributions on four variables. The factorization (2.1) defines a family of distributions. By selecting suitable conditional probability distribution functions $p(y_i|y_{\mathrm{pa}_G(i)})$ and *parameterizing* them with a set of parameters $w \in \mathbb{R}^d$ to obtain $p(y_i|y_{\mathrm{pa}_G(i)}; w)$ we can identify members of this family. In later sections we will discuss parameterization and parameter learning in detail.

The most popular graphical models we will concentrate on in this monograph are *undirected graphical models*, also known as Markov random fields (MRF). More information about directed models can be found in [9, 77].

A MRF defines a family of joint probability distributions by means of an undirected graph $G = (V, \mathcal{E})$ as factorization

$$p(y) = \frac{1}{Z} \prod_{C \in \mathcal{C}(G)} \psi_C(y_C), \tag{2.2}$$

where $\mathcal{C}(G)$ denotes the set of all *cliques*[1] of G. By Y_C we denote the subset of variables that are indexed by C. The normalizing constant Z is given by

$$Z = \sum_{y \in \mathcal{Y}} \prod_{C \in \mathcal{C}(G)} \psi_C(y_C) \tag{2.3}$$

[1] Given $G = (V, \mathcal{E})$, a subset $W \subseteq V$ of the vertices is a clique if for any $i, j \in W$ we have $\{i, j\} \subseteq \mathcal{E}$, that is there exist an edge for any pair of vertices in W.

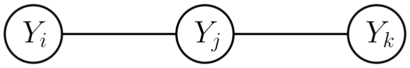


Fig. 2.2 A Markov random field defining the factorization $p(y) = \frac{1}{Z}\psi_i(y_i)\psi_j(y_j)\psi_l(y_l)$ $\psi_{i,j}(y_i,y_j)\psi_{j,k}(y_j,y_k)$.

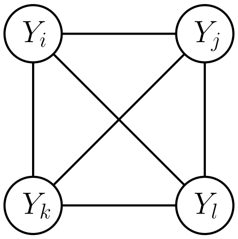


Fig. 2.3 A Markov random field with completely connected graph, defining the factorization $p(y) = \frac{1}{Z}\prod_{A\in 2^{\{i,j,k,l\}}}\psi_A(y_A)$.

and is known as *partition function*. The partition function computes a sum over all configurations $\mathcal{Y} = \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_{|V|}$. The functions $\psi_C\colon\mathcal{Y}_C \to \mathbb{R}_+$ are the so called *potential functions* or *factors*. Each factor ψ_C defines an interaction between one or more variables but in contrast to Bayesian networks it is not a conditional probability but an arbitrary non-negative function. Two examples of MRF and their factorization are shown in Figures 2.2 and 2.3.

The relation between the factorization (2.2) and G is well-defined but cumbersome: imagine we would like to use only pairwise interactions between all pairs of variables shown in the graph of Figure 2.3. To obtain a form (2.2) that contains all pairwise potentials we would need to specify G as shown in Figure 2.3 but set most factors $\psi_C(Y_C) = 1$ for all $|C| < 2$ and $|C| > 2$. The reason for this inefficiency is that the graph that defines the Markov random field does not make explicit the factorization.

A more convenient graphical model to directly specify a factorization is a *factor graph* as introduced in the following section.

2.1 Factor Graphs

Factor graphs are undirected graphical models that make explicit the factorization of the probability function [85]. We define a factor graph as follows.

Definition 2.1 (Factor graph). A *factor graph* is a tuple $(V,\mathcal{F},\mathcal{E})$ consisting of a set V of *variable nodes*, a set $\mathcal{F}$ of *factor nodes*, and a set $\mathcal{E} \subseteq V \times \mathcal{F}$ of edges having one endpoint at a variable node and the other at a factor node. Let $N\colon \mathcal{F} \to 2^V$ be the *scope* of a factor, defined as the set of neighboring variables,

$$N(F) = \{i \in V\colon (i,F) \in \mathcal{E}\}. \tag{2.4}$$

Then the factor graph defines a family of distributions that factorize according to

$$p(y) = \frac{1}{Z} \prod_{F \in \mathcal{F}} \psi_F(y_{N(F)}), \tag{2.5}$$

with

$$Z = \sum_{y \in \mathcal{Y}} \prod_{F \in \mathcal{F}} \psi_F(y_{N(F)}). \tag{2.6}$$

By convention, when drawing a factor graph, factor nodes are drawn as "■" and variable nodes are drawn as "○." The edges are drawn as undirected edges between variable and factor nodes. Let us write the shorthand $y_F := y_{N(F)}$ from now on.

Two examples of factor graphs are shown in Figures 2.4 and 2.5.

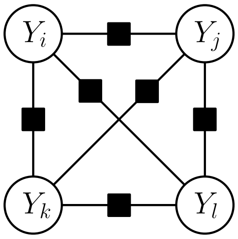


Fig. 2.4 A possible factorization represented by the Markov random field in Figure 2.3. Here only pairwise interactions are used.

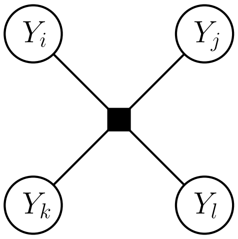


Fig. 2.5 Another possible factorization represented by the Markov random field in Figure 2.3. The MRF specification of a family of distributions cannot distinguish between the factorization in Figures 2.4 and 2.5.

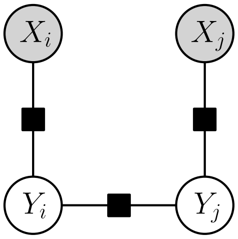


Fig. 2.6 A factor graph specifying a conditional distribution $p(Y_i = y_i, Y_j = y_j | X_i = x_i, X_j = x_j) = \frac{1}{Z(x_i,x_j)} \psi_i(y_i;x_i)\psi_j(y_j;x_j)\psi_{i,j}(y_i,y_j)$.

2.1.1 Conditional Distributions: Conditional Random Fields

We often have access to measurements that correspond to variables that are part of the model. In that case we can directly model the *conditional* distribution $p(Y = y|X = x)$, where $X = x$ is the observation that is always available.

This can be expressed compactly using conditional random fields (CRF) with the factorization provided by a factor graphs as shown in Figure 2.6. The observations $X = x$ we condition on are drawn as shaded variable nodes and the respective factors have access to the values of the observation variables they are adjacent to. Then (2.5) becomes

$$p(Y = y|X = x) = \frac{1}{Z(x)} \prod_{F \in \mathcal{F}} \psi_F(y_F; x_F), \tag{2.7}$$

with

$$Z(x) = \sum_{y \in \mathcal{Y}} \prod_{F \in \mathcal{F}} \psi_F(y_F; x_F). \tag{2.8}$$

Note that the normalizing constant $Z(x)$ in (2.7) now depends on the observation. Given the distribution (2.7) we typically would like to infer *marginal* probabilities $p(Y_F = y_F | x)$ for some factors $F \in \mathcal{F}$. For example, the marginal probability $p(Y_i = \text{“foreground”} | x)$ in an image segmentation model could mean the marginal probability that a pixel i is labeled foreground, given the observed image. In a later section we discuss conditional random fields and inference problems in detail.

2.2 Energy Minimization and Factor Graphs

In computer vision the term *energy minimization* is popularly used to describe approaches in which the solution to the problem is determined by minimizing a function, the “energy.” The energy function is defined for all feasible solutions and measures the quality of a solution.[2]

We now show how energy minimization is interpreted as solving for the state of maximum probability in (2.5).

We define an energy function for each factor $F \in \mathcal{F}$,

$$E_F : \mathcal{Y}_{N(F)} \to \mathbb{R}, \tag{2.9}$$

where $\mathcal{Y}_F = \bigtimes_{i \in N(F)} \mathcal{Y}_i$ is the product domain of the variables adjacent to F. We define the factors $\psi_F : \mathcal{Y}_F \to \mathbb{R}_+$ and energy function as

$$\psi_F(y_F) = \exp(-E_F(y_F)), \quad \text{and} \quad E_F(y_F) = -\log(\psi_F(y_F)). \tag{2.10}$$

We can now rewrite $p(Y)$ in (2.5) as follows.

$$p(Y = y) = \frac{1}{Z} \prod_{F \in \mathcal{F}} \psi_F(y_F) \tag{2.11}$$

$$= \frac{1}{Z} \prod_{F \in \mathcal{F}} \exp(-E_F(y_F)) \tag{2.12}$$

[2] The term “energy” originates with statistical mechanics. For a fascinating account of the connections between physics, computer science and information theory, see [104].

$$= \frac{1}{Z} \exp\left(-\sum_{F \in \mathcal{F}} E_F(y_F)\right), \tag{2.13}$$

with the normalizing constant taking the form:

$$Z = \sum_{y \in \mathcal{Y}} \exp\left(-\sum_{F \in \mathcal{F}} E_F(y_F)\right). \tag{2.14}$$

Finding the state $y \in \mathcal{Y}$ with the highest probability can now be seen as an energy minimization problem:

$$\begin{aligned}
\operatorname*{argmax}_{y \in \mathcal{Y}} p(Y = y) &= \operatorname*{argmax}_{y \in \mathcal{Y}} \frac{1}{Z} \exp\left(-\sum_{F \in \mathcal{F}} E_F(y_F)\right) && (2.15)\\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} \exp\left(-\sum_{F \in \mathcal{F}} E_F(y_F)\right) && (2.16)\\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} -\sum_{F \in \mathcal{F}} E_F(y_F) && (2.17)\\
&= \operatorname*{argmin}_{y \in \mathcal{Y}} \sum_{F \in \mathcal{F}} E_F(y_F). && (2.18)
\end{aligned}$$

Energy minimization approaches are a success story in computer vision and often the most efficient technique to solve a problem. The probabilistic interpretation (2.13) differs in two ways,

(1) it provides a natural way to quantify prediction uncertainty by means of marginal distributions $p(Y_F)$, and
(2) it enables parameter learning by the maximum likelihood principle.

In the last decade breakthroughs have been made in enabling similar advantages for models in which only energy minimization is tractable. We will discuss these advances in detail in the later section *Structured Support Vector Machines.*

Let us now introduce parameterization of factor graphs, followed by a description of the key inference and learning tasks on factor graphs.

2.3 Parameterization

Factor graphs define a family of distributions. By introducing *parameters* into the model we can identify members of this family with parameter values. *Parameterization* is the process of deciding what parameters should be used, how they are shared across different parts of the model, and how they are used to produce the energies for each factor. A high-level illustration of what is achieved by parameterizing the model is shown in Figure 2.7.

Parameters are typically introduced into the energy function as follows. For a given factor $F \in \mathcal{F}$ and parameter vector $w \in \mathbb{R}^D$ we define $E_F : \mathcal{Y}_{N(F)} \times \mathbb{R}^D \to \mathbb{R}$, and write $E_F(y_F; w)$ to show the dependence on w. As an example consider binary image segmentation with a pairwise energy function encouraging nearby pixels to take the same value, taking the form:

$$E_F : \{0,1\} \times \{0,1\} \times \mathbb{R}^2 \to \mathbb{R}, \tag{2.19}$$

with

$$E_F(0,0;w) = E_F(1,1;w) = w_1, \tag{2.20}$$

$$E_F(0,1,w) = E_F(1,0,w) = w_2, \tag{2.21}$$

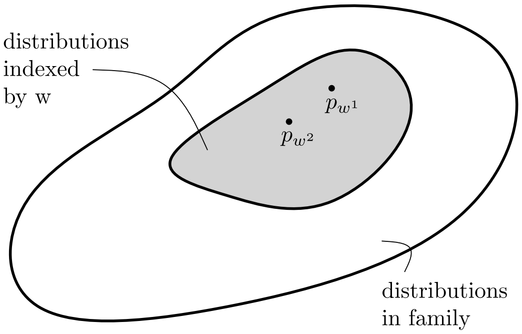


Fig. 2.7 Schematic relationship of the families of distributions: the full set of all distributions in the family defined by a factor graph (shown in white) is restricted to a subset (shown in gray) by parameterization. This subset is indexed by $w \in \mathcal{R}^D$ and any particular choice of $w^1, w^2 \in \mathcal{R}^D$ produces one probability distribution p_{w^1}, p_{w^2}. Increasing the number of parameters and features enlarges the realizable subset of distributions; decreasing the number of parameters by parameter sharing makes the set smaller.

where w_1 and w_2 are parameters directly describing the energies for the subsets $\{(0,0),(1,1)\}$ and $\{(0,1),(1,0)\}$ of states, respectively.

As discussed in Section 2.1.1 we may let E_F depend on observations x_F. Then, a popular form to parameterize E_F is to use a *linear* function

$$E_F(y_f;x_f,w) = \langle w(y_F),\varphi_F(x_F)\rangle, \tag{2.22}$$

where $\varphi_F\colon\mathcal{X}_F \to \mathbb{R}^D$ is a *feature function* operating on the observations and $w\colon\mathcal{Y}_F \to \mathbb{R}^D$ is a concatenation of weight vectors $w(y_F) \in \mathbb{R}^D$, one for each $y_F \in \mathcal{Y}_F$. In the above binary image segmentation task, we might use a unary energy function for each pixel, where $\varphi_F(x_F)$ extracts D-dimensional image features of the image patch observation x_F and $w(0), w(1) \in \mathbb{R}^D$ are learnable weight vectors.

2.4 Inference and Learning Tasks

Once a factor graph model has been fully specified and parameterized there are two tasks left to do: to *learn* its parameters, for example from training instances, and to use the model for solving *inference* tasks on future data instances. We now define the different types of learning and inference problems that we will discuss in this monograph, starting with inference.

2.4.1 Inference Tasks

Ultimately, the goal of probabilistic modeling in computer vision is to make predictions about unobserved properties for a given data instance. Obviously, we would like these predictions to be *as good as possible*, which shows the necessity for a measure of *prediction quality.*

We formalize this in the framework of *statistical decision theory* [19]: let $d(X,Y)$ denote the probability distribution of the data for the problem we try to solve, which we factor into the conditional probability distribution of the label $d(y|x)$, and a data prior $d(x)$. Furthermore, let $\Delta\colon\mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ be a *loss function* where $\Delta(y,y')$ specifies the cost of predicting y' for a sample when the correct label is y. For any sample $x \in \mathcal{X}$ and prediction rule $f : \mathcal{X} \to \mathcal{Y}$ we can measure the quality of predicting the label $f(x)$ by the *expected loss* of this decision

$$\begin{aligned}\mathcal{R}_f^{\Delta}(x) &= \mathbb{E}_{y\sim d(y|x)}\,\Delta(y,f(x)) && (2.23)\\ &= \sum_{y\in\mathcal{Y}} d(y|x)\,\Delta(y,f(x)). && (2.24)\end{aligned}$$

Assuming that the parameterized distribution $p(y|x,w)$ reflects $d(y|x)$ well, we obtain the following approximation

$$\begin{aligned}\mathcal{R}_f^{\Delta}(x) &\approx \mathbb{E}_{y\sim p(y|x,w)}\,\Delta(y,f(x)) && (2.25)\\ &= \sum_{y\in\mathcal{Y}} p(y|x,w)\,\Delta(y,f(x)). && (2.26)\end{aligned}$$

The two important inference problems frequently encountered in computer vision applications, *maximum aposteriori* (MAP) inference and *probabilistic inference*, can be interpreted as the optimal decision rules for two specific loss functions.

Arguably the most common loss function for classification tasks is the 0/1 loss, $\Delta_{0/1}(y,y') = [y \neq y']$, i.e., $\Delta(y,y') = 0$ for $y = y'$, and $\Delta(y,y') = 1$ otherwise. Computing its expected loss we obtain

$$\mathcal{R}_f^{0/1}(x) = 1 - p(Y = f(x)|x,w). \qquad (2.27)$$

This expression is minimized by choosing $f(x) = \operatorname{argmax}_{y\in\mathcal{Y}}$ $p(y|x,w)$ for every $x \in \mathcal{X}$, which shows that the optimal prediction rule in this case is MAP inference.

Problem 2.1(Maximum A Posteriori (MAP) Inference). Given a factor graph, parameterization, and weight vector w, and given the observation x, find the state $y^* \in \mathcal{Y}$ of maximum probability,

$$y^* = \underset{y\in\mathcal{Y}}{\operatorname{argmax}}\; p(Y = y|x,w) = \underset{y\in\mathcal{Y}}{\operatorname{argmax}}\; E(y;x,w). \qquad (2.28)$$

Another popular choice of loss function for structured prediction tasks is the *Hamming loss*: $\Delta(y,y')_H = \frac{1}{|V|}\sum_{i\in V}[y_i \neq y'_i]$. It is sometimes more intuitive than the 0/1-loss. For example, in pixel-wise image segmentation, the Hamming loss is proportional to the number of misclassified pixels, whereas the 0/1-loss assigns the same cost to every labeling that is not pixel-by-pixel identical to the intended one. For the Hamming loss, the expected loss takes the form

$$\mathcal{R}_f^H(x) = 1 - \frac{1}{|V|} \sum_{i \in V} p(Y_i = f(x)_i | x, w), \tag{2.29}$$

which is minimized by predicting with $f(x)_i = \operatorname{argmax}_{y_i \in \mathcal{Y}_i} p(Y_i = y_i | x, w)$, that is, by maximizing the marginal distribution of each node. To evaluate this prediction rule, we rely on *probabilistic inference*.

Problem 2.2 (Probabilistic Inference). Given a factor graph, parameterization, and weight vector w, and given the observation x, find the value of the log partition function and the marginal distributions for each factor,

$$\log Z(x,w) = \log \sum_{y \in \mathcal{Y}} \exp(-E(y;x,w)), \tag{2.30}$$

$$\mu_F(y_F) = p(Y_F = y_F | x, w), \quad \forall F \in \mathcal{F}, \forall y_F \in \mathcal{Y}_F. \tag{2.31}$$

When using the Hamming loss in conjunction with a distribution $p(y|x,w)$, we use the per-variable marginal distributions $p(Y_i = y_i|x,w)$ to make a single joint prediction y for all variables. This produces a small loss in expectation, but might produce a labeling y that does not have a low energy. Furthermore, if we allow infinite energy values in the factors — in effect making some configurations infeasible — then a labeling produced from variable marginals can still have an infinite energy.

Comparing the two inference problems, the MAP inference provides us with the *mode* of $p(y|x,w)$, whereas the result μ of the probabilistic inference describes the *marginal distributions* of $p(y|x,w)$ for each factor. Both inference problems are known to be NP-hard for general graphs and factors [138] but can be tractable if suitably restricted. We will discuss these issues and the use of both (2.28) and (2.31) in computer vision in later parts.

Example 2.1 (Probabilistic inference and MAP). To illustrate how the two inference methods differ qualitatively, in Figure 2.8 to 2.11 we visualize the results of probabilistic inference and MAP inference on

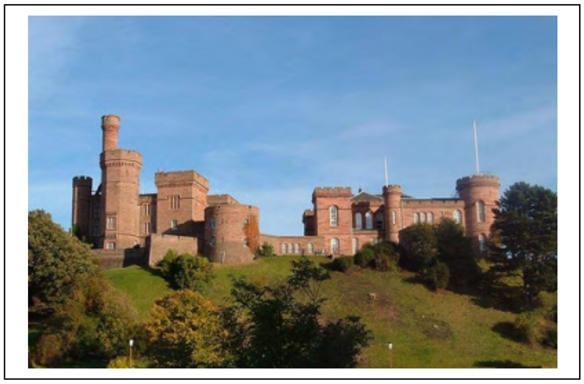


Fig. 2.8 Test image x to be classified, 384 × 256 pixels. (Image source: Dave Conner, `http://www.flickr.com/photos/conner395/2182057020/`)

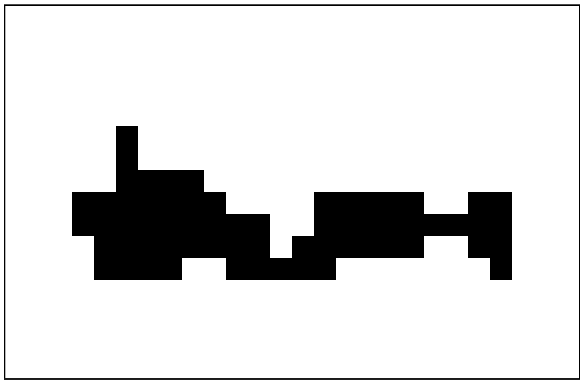


Fig. 2.9 Ground truth annotation (24 × 16 blocks of 16 × 16 pixels). The ground truth annotation of 108 images is used to learn a 2400-dimensional parameter vector w.

the task of recognizing man-made objects in images, originally proposed in [87].

We have one binary variable Y_i per 16×16 block i of pixels. The marginal distribution $p(y_i|x,w)$ of the block predicts the presence of a man-made structure. This marginal distribution is visualized in Figure 2.10. The most likely labeling — that is, the *mode of the distribution* — is shown in Figure 2.11. The model is a factor graph consisting of two different types of factors and has a total of 2,400 parameters that are learned from 108 annotated training images.

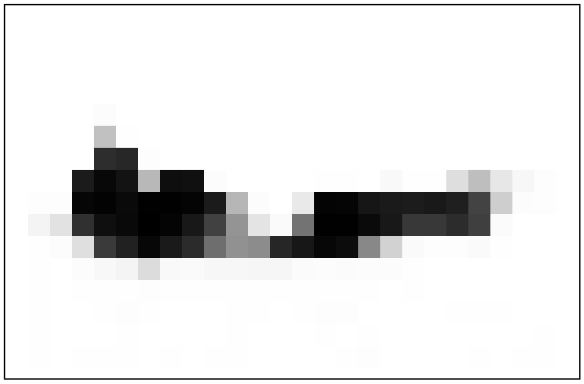


Fig. 2.10 Visualization of the marginal distribution $p(Y_i = 1|x,w)$, where $Y_i = 1$ denotes a man-made structure. Probabilities close to one are dark, probabilities close to zero are white.

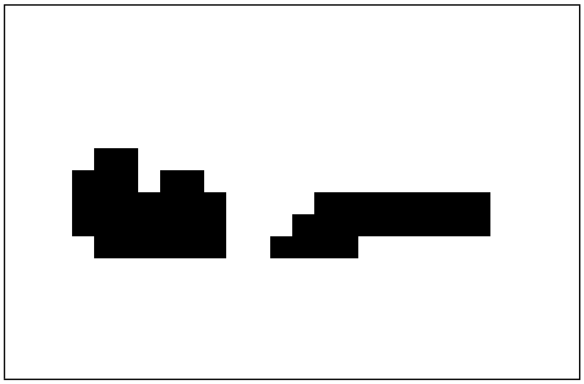


Fig. 2.11 Visualization of the MAP prediction $y^* = \operatorname{argmax}_{y \in \mathcal{Y}} p(Y = y|x,w)$.

We now define the learning problem but postpone a detailed description of how to train CRF and general factor graph models to a later part of this monograph.

2.4.2 Learning Tasks

Learning graphical models from training data is a way to find among a large class of possible models a single one that is best is some sense for the task at hand. In general, the notion of *learning* in graphical models is slightly ambiguous because each part of a graphical model — random variables, factors, and parameters — can in principle be learned from data. In this monograph, as in most computer vision applications, we assume that the model structure and parameterization are specified manually, and learning amounts to finding a vector of real-valued parameters. We will also only consider *supervised learning*, that is we assume a set $\{(x^n, y^n)\}_{n=1,\dots,N}$ of fully observed independent and identically distributed (*i.i.d.*) instances from the true data distribution $d(X,Y)$ to be given to us, which we will use to determine w^*.

Problem 2.3 (Probabilistic Parameter Learning). Let $d(y|x)$ be the (unknown) conditional distribution of labels for a problem to be solved. For a parameterized conditional distribution $p(y|x,w)$ with parameters $w \in \mathbb{R}^D$, *probabilistic parameter learning* is the task of finding a point estimate of the parameter w^* that makes $p(y|x,w^*)$ closest to $d(y|x)$ for every $x \in \mathcal{X}$.

If for a specific prediction task we know which loss function Δ to use at prediction time, and if we decide on MAP-prediction as the decision rule we will follow, we can alternatively aim at learning by *expected loss minimization.*

Problem 2.4 (Loss-Minimizing Parameter Learning). Let $d(x,y)$ be the unknown distribution of data in labels, and let $\Delta\colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ be a loss function. *Loss minimizing parameter learning* is the task of finding a parameter value w^* such that the expected prediction risk

$$\mathbb{E}_{(x,y)\sim d(x,y)}[\Delta(y, f_p(x))] \tag{2.32}$$

is as small as possible, where $f_p(x) = \operatorname{argmax}_{y\in\mathcal{Y}} p(y|x,w^*)$.

We will describe approaches to solve both learning problems in *Probabilistic Parameter Learning* and *Prediction-based Parameter Learning*. The first is *probabilistic parameter learning* based on the principle of maximum likelihood estimation. The second is *prediction-based parameter learning* here only the MAP predictions of the model are used to learn the parameters.

3

Inference in Graphical Models

We now discuss popular methods used to solve the probabilistic inference problem in discrete factor graph models. For general factor graphs this problem is known to be NP-hard, but for graphs that do not contain cycles the problem can be solved efficiently.

For graphs with cycles a number of *approximate inference methods* have been proposed that provide approximate answers. These approximate inference methods can be divided into two groups, deterministic approximations which are solved exactly, and Monte Carlo based approximations.

We start to explain the popular *belief propagation* method which is exact for acyclic graphs and provides an approximation in the general case.

3.1 Belief Propagation and the Sum-Product Algorithm

The sum–product algorithm [85, 98] is a dynamic programming algorithm that given a discrete distribution in the form (2.5) computes the normalizing constant Z and the marginal distributions $p(y_i)$ and $p(y_F)$ for all variables $i \in V$ and factors $F \in \mathcal{F}$, respectively.

The original algorithm is defined for tree-structured[1] factor graphs and provides the exact solution. When the algorithm is modified to work with general factor graphs it is known as *loopy belief propagation* and it provides only an *approximation* to Z and the marginals. We first discuss the exact algorithm and then the extension to the general case. For numerical reasons we describe the algorithm in terms of log-factors, i.e., by using $\log \psi_F = -E_F$. We will discuss implementation caveats later.

The way the algorithm works is by computing vectors termed "messages" for each edge in the factor graph. In particular, each edge $(i,F) \in \mathcal{E}$ has two such vectors associated with it,

(1) $q_{Y_i \to F} \in \mathbb{R}^{\mathcal{Y}_i}$, the variable-to-factor message, and
(2) $r_{F \to Y_i} \in \mathbb{R}^{\mathcal{Y}_i}$, the factor-to-variable message.

When the algorithm has finished computing all these messages, then the quantities of interest — Z and the marginal distributions $p(y_i)$ and $p(y_F)$ — are simple to compute.

Let us first take a look how each individual message is computed, then we will discuss the order of computing messages.

3.1.1 Computing the Messages

For computing the variable-to-factor message, we define the set $M(i)$ of factors adjacent to variable i as

$$M(i) = \{F \in \mathcal{F} \colon (i,F) \in \mathcal{E}\}, \tag{3.1}$$

similar to how $N(F)$ has been defined as set of variables adjacent to F. Then, the variable-to-factor message is computed as

$$q_{Y_i \to F}(y_i) = \sum_{F' \in M(i) \setminus \{F\}} r_{F' \to Y_i}(y_i). \tag{3.2}$$

This computation is visualized in Figure 3.1: the incoming message vectors $r_{F' \to Y_i}$, except the message $r_{F \to Y_i}$ coming from F, are added to yield the outgoing message $q_{Y_i \to F}$. If there is no term in the summation the message is simply the all-zero vector.

[1] A factor graph is *tree-structured* if it is connected and does not contain a cycle.

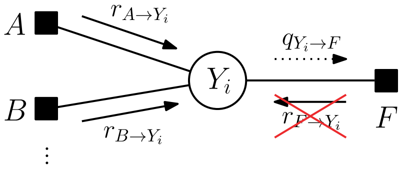


Fig. 3.1 Visualization of the computation of the variable-to-factor message $q_{Y_i\to F}$ by Equation (3.2).

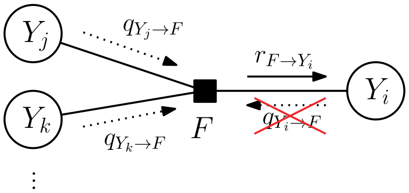


Fig. 3.2 Visualization of the computation of the factor-to-variable message $r_{F\to Y_i}$ by Equation (3.3).

Likewise the factor-to-variable message is computed as follows.

$$r_{F\to Y_i}(y_i) = \log \sum_{\substack{y'_F\in\mathcal{Y}_F,\\ y'_i=y_i}} \exp\left(-E_F(y'_F) + \sum_{j\in N(F)\setminus\{i\}} q_{Y_j\to F}(y'_j)\right). \tag{3.3}$$

Figure 3.2 illustrates the computation: the incoming variable-to-factor messages from every edge except the one we are computing the message for are combined to yield the outgoing factor-to-variable message. Here the computation (3.3) is more complex than (3.2) because we additionally sum over the states of all adjacent variables.

3.1.2 Message Ordering

The Equations (3.2) and (3.3) for computing the messages depend on previously computed messages. The only messages that do not depend on previous computation are the following.

- The variable-to-factor messages in which no other factor is adjacent to the variable; then the summation in (3.2) is empty and the message will be zero.

- The factor-to-variable messages in which no other variable is adjacent to the factor; then the inner summation in (3.3) will be empty.

For tree-structured factor graphs there always exist at least one such message that can be computed initially. The computed message in turn enables the computation of other messages. Moreover, we can order all message computations in such a way that we resolve all dependencies and eventually have computed all messages.

For tree-structured graphs this corresponds to the scheme shown in Figures 3.3 and 3.4. We first designate an arbitrary variable node — here we chose Y_m — as the "tree root." Then we compute all messages directed *toward the root*, starting with the leaf nodes of the factor graph because these are the only messages we can initially compute. We compute the remaining messages in an order that follows the leaf-to-root structure of the tree. Let us therefore call this step the "leaf-to-root" phase. From Figure 3.3 it is clear that for each message computation we will always have previously computed the information it depends upon.

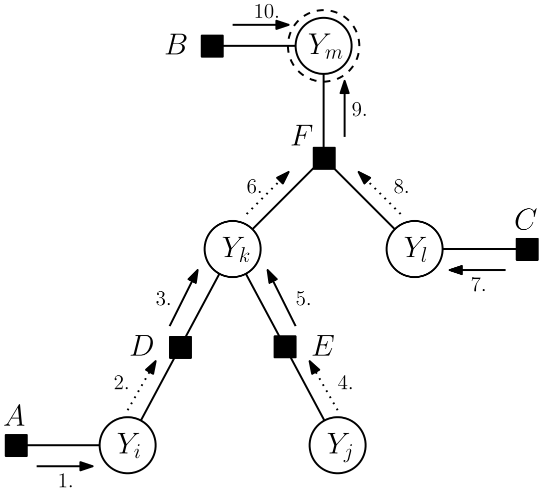


Fig. 3.3 One possible leaf-to-root message schedule in the sum–product algorithm. Factor-to-variable messages are drawn as arrows, variable-to-factor messages as dotted arrows. The tree is rooted in Y_m and the node is marked with a dashed circle.

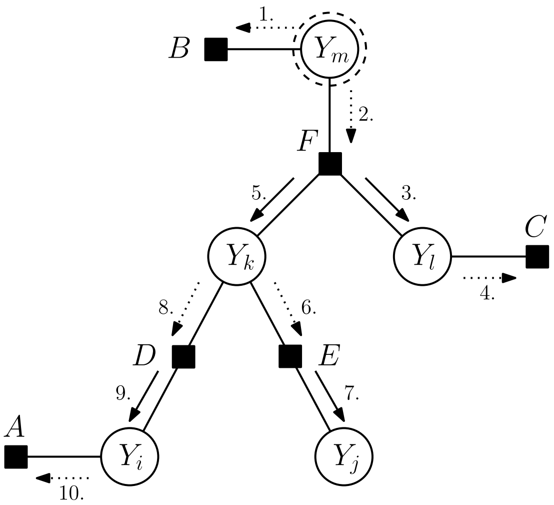


Fig. 3.4 The root-to-leaf message schedule, the reverse of the schedule shown in Figure 3.3. Note that all edges now pass the message of type opposite of what they passed in the leaf-to-root phase. Thus, for each edge both message types are available.

Once we have reached the root Y_m we reverse the schedule as shown in Figure 3.4 and again we are sure to previously have computed the information the message depends on. When we terminate this "root-to-leaf" phase we have finished computing all messages. Moreover we have not performed any duplicate computation. Let us now discuss how to compute the partition function and marginals from the messages.

3.1.3 Computation of the Partition Function and Marginals

The partition function Z can be computed as soon as the leaf-to-root phase is finished. Because the value of Z can become very large we usually work with the log-partition function $\log Z$. We can compute it by summing all factor-to-variable messages directed to the tree root Y_r as follows.

$$\log Z = \log \sum_{y_r \in \mathcal{Y}_r} \exp \left(\sum_{F \in M(r)} r_{F \to Y_r}(y_r) \right). \tag{3.4}$$

Once we have completed the root-to-leaf phase we can use the messages and the computed value of $\log Z$ to compute marginals for factors and variables as follows. The factor marginals take the form:

$$\mu_F(y_F) = p(Y_F = y_F) = \exp\left(-E_F(y_F) + \sum_{i \in N(F)} q_{Y_i \to F}(y_i) - \log Z\right), \tag{3.5}$$

for each factor $F \in \mathcal{F}$ and state $y_F \in \mathcal{Y}_F$. Equation (3.5) is visualized in Figure 3.5. Likewise, the variable marginals are computed as

$$p(Y_i = y_i) = \exp\left(\sum_{F \in M(i)} r_{F \to Y_i}(y_i) - \log Z\right), \tag{3.6}$$

for each variable Y_i and value $y_i \in \mathcal{Y}_i$. The computation is visualized in Figure 3.6.

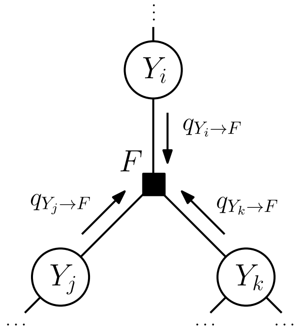


Fig. 3.5 Visualization of the computation of the marginal distribution at factor node F, Equation (3.5).

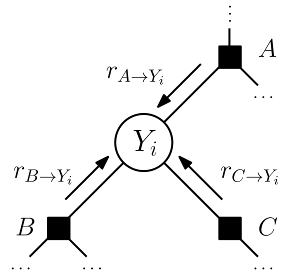


Fig. 3.6 Visualization of the computation of the marginal distribution at variable Y_i, Equation (3.6).

3.1.4 Implementation Caveats

Having explained all the necessary ingredients, we are now able to give the full belief propagation algorithm for tree-structured factor graphs, shown in Algorithm 1 on page 212.

The algorithm is straight-forward to implement but requires some care when computing (3.3), (3.4), (3.5), and (3.6): the naive computation of the log–sum–exp expression with large numbers leads to numerical instabilities that quickly amplify across the recursive computation. This problem is mitigated by using the identity

$$\log \sum_i \exp(v_i) = \alpha + \log \sum_i \exp(v_i - \alpha), \tag{3.7}$$

for all $\alpha \in \mathbb{R}$. By setting $\alpha = \max_i v_i$ for each log–sum–exp expression in (3.3)–(3.6) and using the right-hand side of the identity to evaluate the left hand side expression we obtain a numerically stable method.

Example 3.1 (Pictorial Structures). The *pictorial structures* model first proposed by Fischler and Elschlager [40] models objects as individual parts with pairwise relations between parts. For each part an appearance model is used to evaluate an energy for each possible position that part can take. The pairwise relations between parts encourage parts to take pairwise likely configurations.

The example in Figures 3.7 and 3.8 show a pictorial structures model for a person taken from [37]. It uses eleven different body parts and the pairwise relations roughly follow the skeletal structure of the human body: the left and right arm as well as the left and right leg are connected to the torso.

In order to use the model, we need to provide an appearance model in the form of an energy $E_{F^{(1)}_{\text{top}}}(y_{\text{top}};x)$ for each body part (top, head, torso, etc.), where x is the observed image and y takes labels from a large discretized pose space. In [37] the pose space is a four tuple (x,y,s,θ), where (x,y) are the absolute image coordinates, s is a scale, and θ is the rotation of the part. Furthermore we need to provide pairwise energies $E_{F^{(2)}_{\text{top,head}}}(y_{\text{top}},y_{\text{head}})$ encoding which pairwise relations between pairs of parts are preferred.

Algorithm 1: Belief Propagation on Trees

```
$(\log Z, \mu) = \textsc{BeliefPropagation}(V, \mathcal{F}, \mathcal{E}, E)$
Input:
  $(V, \mathcal{F}, \mathcal{E})$, tree-structured factor graph,
  $E$, energies $E_F$ for all $F \in \mathcal{F}$.
Output:
  $\log Z$, log partition function of $p(y)$,
  $\mu$, marginal distributions $\mu_F$ for all $F \in \mathcal{F}$.
Algorithm:
Fix an element of $V$ arbitrarily as tree root
Compute leaf-to-root order $R$ as sequence of directed
  edges of $\mathcal{E}$
for $i = 1, \ldots, |R|$ do
  if $(v, F) = R(i)$ is variable-to-factor edge then
    Compute $q_{Y_i \to F}$ using (3.2)
  else
    $(F, v) = R(i)$ is factor-to-variable edge
    Compute $r_{F \to Y_i}$ using (3.3)
  end if
end for
Compute $\log Z$ by (3.4)
Compute root-to-leaf order $R' = \text{reverse}(R)$
for $i = 1, \ldots, |R'|$ do
  if $(v, F) = R'(i)$ is variable-to-factor edge then
    Compute $q_{Y_i \to F}$ using (3.2)
    Compute $\mu_F$ using (3.5)
  else
    $(F, v) = R'(i)$ is factor-to-variable edge
    Compute $r_{F \to Y_i}$ using (3.3)
    Compute $p(y_i)$ using (3.6)
  end if
end for
```

Given a new test image, we can use the belief propagation algorithm for tree-structured graphs to perform inference over the model in time

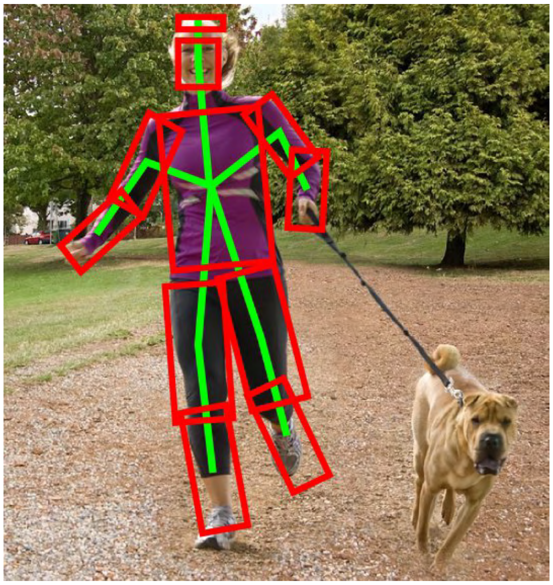


Fig. 3.7 Visualization of a labeling as produced by pictorial structures (Image source: `http://www.flickr.com/photos/lululemonathletica/3908348636/`).

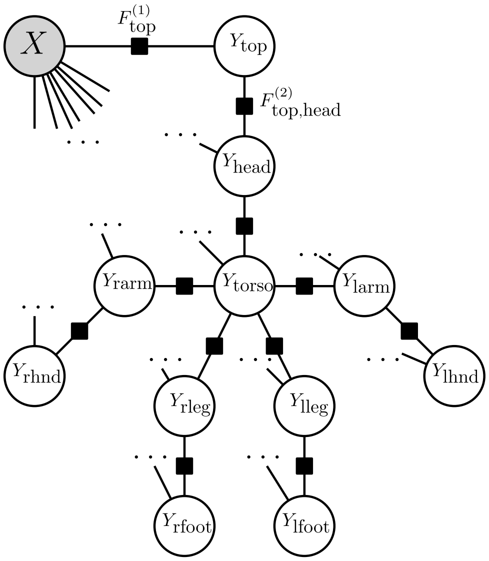


Fig. 3.8 Pictorial structure model of Felzenszwalb and Huttenlocher [37] for person recognition. Each part-variable takes as label a discretized tuple (x, y, s, θ) of position, scale, and rotation states.

$O(kL^2)$, where k is the number of parts and L is the number of possible labels for each part variable. Because the labels arise from discretizing a continuous pose space, L is typically large ($L = 500,000$ in the original model of Felzenszwalb and Huttenlocher) and therefore computing (3.2) or (3.19) is too expensive to be practical.

Felzenszwalb and Huttenlocher [37] make this computation tractable by restricting the pairwise energy functions to be of the form:

$$E_{F^{(2)}_{\mathrm{a,b}}}(y_{\mathrm{a}}, y_{\mathrm{b}}) = \|U_{ab}(y_a) - U_{ba}(y_b)\|, \tag{3.8}$$

where $\|\cdot\|$ is a norm and U_{ij}, U_{ji} are arbitrary maps, mapping the labels into a fixed Euclidean space $\mathbb{R}^n$. With this choice, Felzenszwalb and Huttenlocher show that it is possible to compute (3.2) and (3.19) in $O(L)$, yielding an overall inference complexity of $O(kL)$. Different choices of U_{ab} and norms allow for flexible pose relations. Details can be found in the original paper. Andriluka et al. [5] extend the original model and discuss in detail the training of the model.

3.2 Loopy Belief Propagation

Belief propagation can be applied to tree-structured factor graphs and provides an exact solution to the probabilistic inference problem. When the factor graph is not tree-structured but contains one or more cycles, the belief propagation algorithm is not applicable as no leaf-to-root order can be defined. However, the message Equations (3.2) and (3.3) remain well-defined. Therefore, we can initialize all messages to a fixed value and perform the message updates iteratively in a fixed or random order to perform computations "similar" to the original exact algorithm on trees. The resulting algorithm is named *loopy belief propagation*.

The loopy belief propagation algorithm made approximate inference possible in previously intractable models [46]. The empirical performance was consistently reported to be excellent across a wide range of problems and the algorithm is perhaps the most popular approximate inference algorithm for discrete graphical models. In practise, the algorithm does not always converge. If it fails to converge then the beliefs are a poor approximations to the true marginals. The convergence problem and the encouraging positive results remained poorly understood and the theoretical analysis of loopy belief propagation initially lagged behind until Yedidia et al. [171] showed a close connection between the so called *Bethe free energy* approximation in physics and the loopy belief propagation algorithm [104]. It is now known that if the algorithm converges then it converged to a fix-point of the Bethe free energy and this connection has been fruitfully used to derive a family of similar algorithms [59, 160, 165], some of which can ensure convergence to a fix point.

3.2.1 Computing the Messages and Marginals

Compared to belief propagation on trees the equations used to compute messages in loopy belief propagation change slightly. Whereas the factor-to-variable messages $r_{F \to Y_i}$ are computed as before, the variable-to-factor messages are normalized in every iteration as follows.

$$\bar{q}_{Y_i \to F}(y_i) = \sum_{F' \in M(i) \setminus \{F\}} r_{F' \to Y_i}(y_i), \tag{3.9}$$

$$\delta = \log \sum_{y_i \in \mathcal{Y}_i} \exp(\bar{q}_{Y_i \to F}(y_i)), \tag{3.10}$$

$$q_{Y_i \to F}(y_i) = \bar{q}_{Y_i \to F}(y_i) - \delta. \tag{3.11}$$

The approximate marginals — often named *beliefs* in the loopy belief propagation literature — are computed as before but now a factor-specific normalization constant z_F is used. The factor marginals can be computed at any point in time as follows.

$$\bar{\mu}_F(y_F) = -E_F(y_F) + \sum_{j \in N(F)} q_{Y_j \to F}(y_j), \tag{3.12}$$

$$z_F = \log \sum_{y_F \in \mathcal{Y}_F} \exp(\bar{\mu}_F(y_F)), \tag{3.13}$$

$$\mu_F(y_F) = \exp(\bar{\mu}_F(y_F) - z_F). \tag{3.14}$$

In addition to the factor marginals the algorithm also computes the variable marginals in a similar fashion.

$$\bar{\mu}_i(y_i) = \sum_{F' \in M(i)} r_{F' \to Y_i}(y_i), \tag{3.15}$$

$$z_i = \log \sum_{y_i \in \mathcal{Y}_i} \exp(\bar{\mu}_i(y_i)), \tag{3.16}$$

$$\mu_i(y_i) = \exp(\bar{\mu}_i(y_i) - z_i). \tag{3.17}$$

In the original belief propagation algorithm the exact normalizing constant Z is computed at the tree root and applied as normalization constant throughout the tree. For loopy belief propagation this is not possible because the local normalization constant z_F differs at each factor. Instead, an approximation to the log partition function $\log Z$ is computed from the Bethe free energy interpretation as follows [171].

$$\begin{aligned} \log Z = & \sum_{i \in V} (|M(i)| - 1) \left[\sum_{y_i \in \mathcal{Y}_i} \mu_i(y_i) \log \mu_i(y_i) \right] \\ & - \sum_{F \in \mathcal{F}} \sum_{y_F \in \mathcal{Y}_F} \mu_F(y_F) \left(E_F(y_F) + \log \mu_F(y_F) \right). \end{aligned} \tag{3.18}$$

One possible implementation of loopy belief propagation is Algorithm 2 on page 217. In each main iteration of the algorithm all factor-to-variable messages and all variable-to-factor messages are computed for all edges of the factor graph, as shown in Figures 3.9 and 3.10.

3.2.2 Max-product/Max-sum Algorithm

The belief propagation algorithm can also be used to perform MAP inference. As for probabilistic inference it is exact in case the factor graph is a tree. For graphs with cycles it provides an approximation. The MAP inference version of belief propagation is also known as *max-product algorithm*; the version working directly on the energies as the *max-sum algorithm* and *min-sum algorithm*. We will describe the max-sum version of the algorithm.

The basic idea to derive the max-sum algorithm is to replace the marginalization performed in (3.3) by a *maximization*. Additionally the messages are shifted to prevent numerical problems due to large numbers forming from accumulation of messages, although this is not essential to the algorithm. Together this yields the following messages.

Algorithm 2: Loopy Belief Propagation (sum–product)

```
$(\log Z, \mu) = \textsc{SumProductLoopyBP}(V, \mathcal{F}, \mathcal{E}, E, \varepsilon, T)$
**Input:**
    $(V, \mathcal{F}, \mathcal{E})$, factor graph,
    $E$, energies $E_F$ for all $F \in \mathcal{F}$,
    $\varepsilon$, convergence tolerance,
    $T$, maximum number of iterations.
**Output:**
    $\log Z$, approximate log partition function of $p(y)$,
    $\mu$, approximate marginal distributions $\mu_F$ for all $F \in \mathcal{F}$.
**Algorithm:**
$q_{Y_i \to F}(y_i) \leftarrow 0$, for all $(i, F) \in \mathcal{E}, y_i \in \mathcal{Y}_i$
$\mu_F(y_F) \leftarrow 0$, for all $F \in \mathcal{F}, y_F \in \mathcal{Y}_F$
**for** $t = 1, \ldots, T$ **do**
  **for** $(v, F) \in \mathcal{F}$ **do**
    **for** $y_i \in \mathcal{Y}_i$ **do**
      Compute $r_{F \to Y_i}(y_i)$ using (3.3)
    **end for**
  **end for**
  **for** $(v, F) \in \mathcal{F}$ **do**
    **for** $y_i \in \mathcal{Y}_i$ **do**
      Compute $q_{Y_i \to F}(y_i)$ using (3.9) to (3.11)
    **end for**
  **end for**
  Compute approximate marginals $\mu'$ using (3.12) to (3.17)
  $\mathrm{u} \leftarrow \|\mu' - \mu\|_\infty$ {Measure change in beliefs}
  $\mu \leftarrow \mu'$
  **if** $u \leq \varepsilon$ **then**
    **break** {Converged}
  **end if**
**end for**
Compute $\log Z$ using (3.18)
```

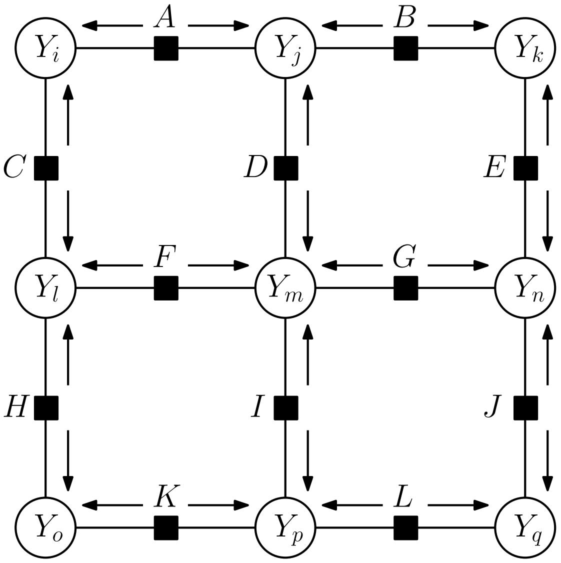


Fig. 3.9 Passing factor-to-variable messages $r_{F \to Y_i}$ by Equation (3.3).

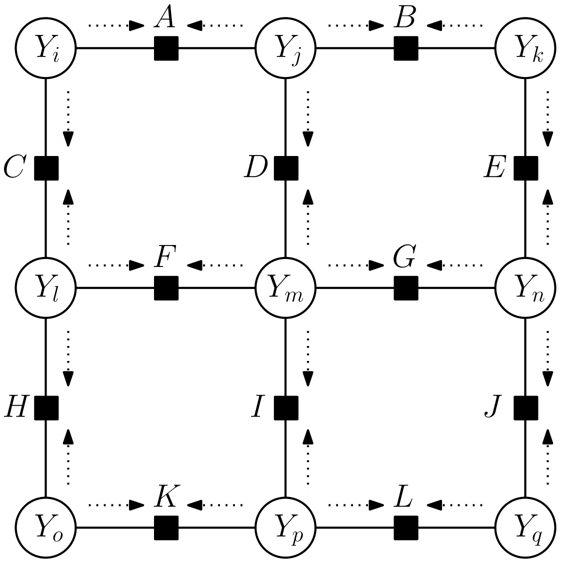


Fig. 3.10 Passing variable-to-factor messages $q_{Y_i \to F}$ by Equations (3.9)–(3.11).

The factor-to-variable message perform maximization over the states of the factor as

$$r_{F\to Y_i}(y_i) = \max_{\substack{y'_F \in \mathcal{Y}_F, \\ y'_i = y_i}} \left(-E_F(y'_F) + \sum_{j \in N(F)\setminus\{i\}} q_{Y_j \to F}(y'_j) \right). \qquad (3.19)$$

The variable-to-factor message are identical to the sum–product version but the normalization moves the mean of the message to zero.

$$\bar{q}_{Y_i\to F}(y_i) = \sum_{F' \in M(i)\setminus\{F\}} r_{F'\to Y_i}(y_i), \qquad (3.20)$$

$$\delta = \frac{1}{|\mathcal{Y}_i|} \sum_{y_i \in \mathcal{Y}_i} \bar{q}_{Y_i\to F}(y_i), \qquad (3.21)$$

$$q_{Y_i\to F}(y_i) = \bar{q}_{Y_i\to F}(y_i) - \delta. \qquad (3.22)$$

The variable max-beliefs are no longer interpretable as marginals but instead $\mu_i(y_i)$ describes the maximum negative energy achievable when fixing the variable to $Y_i = y_i$.

$$\mu_i(y_i) = \sum_{F' \in M(i)} r_{F'\to Y_i}(y_i), \qquad (3.23)$$

To recover a joint minimum energy labeling, we select for each variable Y_i the state $y_i \in \mathcal{Y}_i$ with the maximum max-belief,

$$y_i^* = \operatorname*{argmax}_{y_i \in \mathcal{Y}_i} \mu_i(y_i), \quad \forall i \in V. \qquad (3.24)$$

The overall procedure is shown in Algorithm 3 on page 220. The structure of the algorithm is the same as that of the sum–product version and both the sum–product and max-sum algorithms are typically implemented as one algorithm, differing only in the message updates.

3.2.3 Further Reading on Belief Propagation

For a more detailed review of the belief propagation algorithms and proofs of its correctness for tree-structured factor graphs, see [9, 85, 98, 171]. Alternative message schedules are discussed in [36]. Different message passing algorithms that are guaranteed to converge

Algorithm 3: Loopy Belief Propagation (max-sum)

```
$y^* = \textsc{MaxSumLoopyBP}(V,\mathcal{F},\mathcal{E},E,\varepsilon,T)$
**Input:**
    $(V,\mathcal{F},\mathcal{E})$, factor graph,
    $E$, energies $E_F$ for all $F \in \mathcal{F}$,
    $\varepsilon$, convergence tolerance,
    $T$, maximum number of iterations.
**Output:**
    $y^*$, approximate MAP labeling $y^* \approx \mathrm{argmax}_{y \in \mathcal{Y}}$
    $p(Y = y|x,w)$
**Algorithm:**
$q_{Y_i \to F}(y_i) \leftarrow 0$, for all $(i,F) \in \mathcal{E}, y_i \in \mathcal{Y}_i$
$\mu_i(y_i) \leftarrow \infty$, for all $i \in V, y_i \in \mathcal{Y}_i$
**for** $t = 1,\dots,T$ **do**
  **for** $(v,F) \in \mathcal{F}$ **do**
    **for** $y_i \in \mathcal{Y}_i$ **do**
      Compute $r_{F \to Y_i}(y_i)$ using (3.19)
    **end for**
  **end for**
  **for** $(v,F) \in \mathcal{F}$ **do**
    **for** $y_i \in \mathcal{Y}_i$ **do**
      Compute $q_{Y_i \to F}(y_i)$ using (3.20) to (3.22)
    **end for**
  **end for**
  Compute variable max-beliefs $\mu'_i$ using (3.23)
  u $\leftarrow \max_{i \in V} \|\mu'_i - \mu_i\|_\infty$ {Measure change in max-beliefs}
  $\mu \leftarrow \mu'$
  **if** $u \leq \varepsilon$ **then**
    **break** {Converged}
  **end if**
**end for**
Compute $y^*$ from max-beliefs using (3.24)
```

have been proposed [78, 174], an overview of different variants and recent results is available in [102]. A number of generalizations, including the *expectation propagation* algorithm have been proposed, for a discussion, see [59, 105].

The max-sum version of loopy belief propagation performs well in practice but its behavior remains poorly understood. For this reason alternative algorithms such as tree-reweighted message passing [78], generalized max-product [142], and max-sum diffusion [162] have been proposed. They come with strong convergence guarantees and additionally provide not only an approximate MAP labeling but also a lower bound on the best achievable energy.

An early work using belief propagation in computer vision for super-resolution is [45]. Felzenszwalb and Huttenlocher [38] showed that the efficiency of belief propagation can be increased when using special types of factors. Then the messages (3.3) and (3.19) can be computed using the fast Fourier transform or a specialized maximization algorithm, respectively. Similarly, Potetz [121] has shown how a flexible class of higher-order factors useful for low-level vision — so called *linear constraint nodes* — can be used efficiently within belief propagation.

3.3 Mean Field Methods

For general discrete factor graph models, performing probabilistic inference is hard. *Mean field methods* perform approximate probabilistic inference by searching within a tractable subset of distributions for the distribution which best approximates the original distribution [66, 160].

One way of finding the best approximating distribution is to pose it as an optimization problem over probability distributions: given a distribution $p(y|x,w)$ and a family $\mathcal{Q}$ of tractable distributions $q \in \mathcal{Q}$ on $\mathcal{Y}$, we want to solve

$$q^* = \operatorname*{argmin}_{q \in \mathcal{Q}} D_{\mathrm{KL}}(q(y) \| p(y|x,w)), \qquad (3.25)$$

where D_{KL} is the *Kullback–Leibler divergence* between two probability distributions.

If the set $\mathcal{Q}$ is rich enough to contain a close approximation to $p(y|x,w)$ and we succeed at finding it, then the marginals of q^* will provide a good approximation to the true marginals of $p(y|x,w)$ that are intractable to compute.[2]

Assume that the set $\mathcal{Q}$ is given. Then the relative entropy $D_{\mathrm{KL}}(q(y)\|p(y|x,w))$ can be rewritten as follows.

$$\begin{aligned}
&D_{\mathrm{KL}}(q(y)\|p(y|x,w)) && (3.26)\\
&\quad = \sum_{y\in\mathcal{Y}} q(y)\log\frac{q(y)}{p(y|x,w)} && (3.27)\\
&\quad = \sum_{y\in\mathcal{Y}} q(y)\log q(y) - \sum_{y\in\mathcal{Y}} q(y)\log p(y|x,w) && (3.28)\\
&\quad = -H(q) + \sum_{F\in\mathcal{F}}\sum_{y_F\in\mathcal{Y}_F} \mu_{F,y_F}(q)E_F(y_F;x_F,w)\\
&\qquad + \log Z(x,w), && (3.29)
\end{aligned}$$

where $H(q) = -\sum_{y\in\mathcal{Y}} q(y)\log q(y)$ is the *entropy* of the distribution q and $\mu_{F,y_F} = \sum_{y\in\mathcal{Y},[y]_{N(F)}=y_F} q(y)$ is the marginal distribution of q on the variables $N(F)$. The exact form of this expression depends on the family $\mathcal{Q}$ and we will see an example below for the so called naive mean field approximation. The term $\log Z(x,w)$ is the log partition function of p. Note that this term does not depend on q and therefore it is not necessary to compute $\log Z(x,w)$ in order to minimize $D_{\mathrm{KL}}(q(y)\|p(y|x,w))$.

Suppose we minimized the above expression. By the knowledge that for any distribution q we are guaranteed to have $D_{\mathrm{KL}}(q(y)\|p(y|x,w)) \geq 0$, the so called *Gibbs inequality*, we obtain the following *mean field lower bound* on the log partition function.

$$\log Z(x,w) \geq H(q) - \sum_{F\in\mathcal{F}}\sum_{y_F\in\mathcal{Y}_F} \mu_{F,y_F}(q)E_F(y_F;x_F,w). \qquad (3.30)$$

Therefore, the mean field method provides as inference results not only approximate marginals μ but also a lower bound on the true log partition function. Typically, the larger the approximating family $\mathcal{Q}$, the

[2] The minimization direction $D_{\mathrm{KL}}(q\|p)$ — the so called *information projection* — of the KL-divergence is important: minimizing $D_{\mathrm{KL}}(p\|q)$ — the so called *moment projection* is desirable but intractable. The difference is explained in [105].

stronger this lower bound becomes. Let us take a look at the most popular mean field method, the so called *naive mean field* method.

3.3.1 Naive Mean Field

In naive mean field, we take the set $\mathcal{Q}$ as the set of all factorial distributions, in the form:

$$q(y) = \prod_{i \in V} q_i(y_i). \tag{3.31}$$

This approximating distribution is visualized in Figures 3.11 and 3.12. For factorial distributions the entropy $H(q)$ decomposes as a sum over

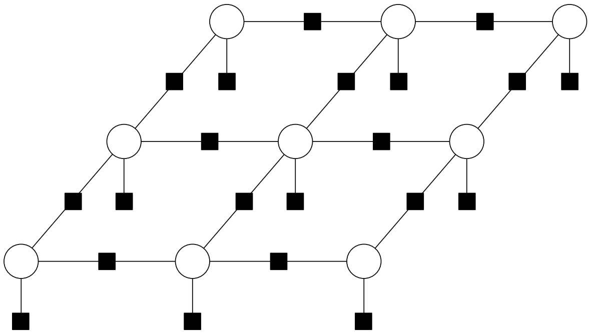


Fig. 3.11 Original intractable factor graph model $p(y)$.

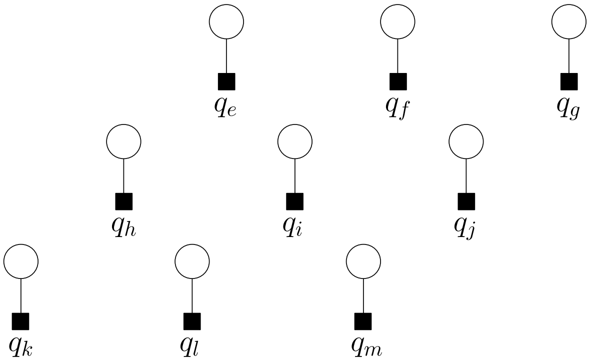


Fig. 3.12 Factorial naive mean field approximation $q(y) = \prod_{i \in V} q_i(y_i)$.

per-variable entropies, i.e.,

$$H(q) = \sum_{i \in V} H_i(q_i) = -\sum_{i \in V} \sum_{y_i \in \mathcal{Y}_i} q_i(y_i) \log q_i(y_i). \tag{3.32}$$

Likewise, the factor marginals $\mu_{F,y_F}(q)$ decompose as the product of the variable marginals with

$$\mu_{F,y_F}(q) = \prod_{i \in N(F)} q_i(y_i). \tag{3.33}$$

Plugging (3.32) and (3.33) into the divergence objective (3.29) yields the following variational inference problem.

$$\begin{aligned}
&\operatorname{argmin}_{q \in \mathcal{Q}} D_{\mathrm{KL}}(q(y) \| p(y|x,w)) && (3.34)\\
&\quad = \operatorname*{argmax}_{q \in \mathcal{Q}} H(q) - \sum_{F \in \mathcal{F}} \sum_{y_F \in \mathcal{Y}_F} \mu_{F,y_F}(q) E_F(y_F; x_F, w) && \\
&\qquad - \log Z(x,w) && (3.35)\\
&\quad = \operatorname*{argmax}_{q \in \mathcal{Q}} H(q) - \sum_{F \in \mathcal{F}} \sum_{y_F \in \mathcal{Y}_F} \mu_{F,y_F}(q) E_F(y_F; x_F, w) && (3.36)\\
&\quad = \operatorname*{argmax}_{q \in \mathcal{Q}} \Bigg[-\sum_{i \in V} \sum_{y_i \in \mathcal{Y}_i} q_i(y_i) \log q_i(y_i) && \\
&\qquad - \sum_{F \in \mathcal{F}} \sum_{y_F \in \mathcal{Y}_F} \left(\prod_{i \in N(F)} q_i(y_i) \right) E_F(y_F; x_F, w) \Bigg]. && (3.37)
\end{aligned}$$

Where we optimize over all $q_i \in \Delta_i$, the probability simplex defined for each $i \in V$ as $q_i(y_i) \geq 0$, and $\sum_{y_i \in \mathcal{Y}_i} q_i(y_i) = 1$. This problem is a maximization problem in which the entropy term is concave and the second term is nonconcave due to products of variables occurring in the expression. Therefore solving this nonconcave maximization problem globally is in general hard. However, when we hold all variables fixed except for a single block $q_i \in \Delta_i$, then we obtain the following tractable concave maximization problem.

$$q_i^* = \operatorname*{argmax}_{q_i \in \Delta_i} \Bigg[- \sum_{y_i \in \mathcal{Y}_i} q_i(y_i) \log q_i(y_i) \\ - \sum_{\substack{F \in \mathcal{F}, \\ i \in N(F)}} \sum_{y_F \in \mathcal{Y}_F} \left(\prod_{j \in N(F) \setminus \{i\}} \hat{q}_j(y_j) \right) q_i(y_i) E_F(y_F; x_F, w) \Bigg], \tag{3.38}$$

where $\hat{q}_j(y_j) = q_j(y_j)$ is held fixed and all constant terms not affected by q_i have been dropped, so that we only need to consider the neighbors of variable i within (3.38). This maximization problem can be analytically solved to obtain the optimal solution q_i^* as

$$q_i^*(y_i) \\ = \exp \left(1 - \sum_{\substack{F \in \mathcal{F}, \\ i \in N(F)}} \sum_{\substack{y_F \in \mathcal{Y}_F, \\ [y_F]_i = y_i}} \left(\prod_{j \in N(F) \setminus \{i\}} \hat{q}_j(y_j) \right) E_F(y_F; x_F, w) + \lambda \right), \tag{3.39}$$

with normalizing constant λ derived as Lagrange multiplier of the constraint $\sum_{y_i \in \mathcal{Y}_i} q_i^*(y_i) = 1$, chosen as the unique value guaranteeing unity,

$$\lambda = -\log \Bigg(\sum_{y_i \in \mathcal{Y}_i} \\ \times \exp \left(1 - \sum_{\substack{F \in \mathcal{F}, \\ i \in N(F)}} \sum_{\substack{y_F \in \mathcal{Y}_F, \\ [y_F]_i = y_i}} \left(\prod_{j \in N(F) \setminus \{i\}} \hat{q}_j(y_j) \right) E_F(y_F; x_F, w) \right) \Bigg), \tag{3.40}$$

where we define the empty product to be 1 in case we have $N(F) \setminus \{i\} = \emptyset$. The quantities involved in updating a single variable distribution are visualized in Figure 3.13. For each $i \in V$ the update of q_i can be carried out efficiently, effectively optimizing (3.37) by block coordinate

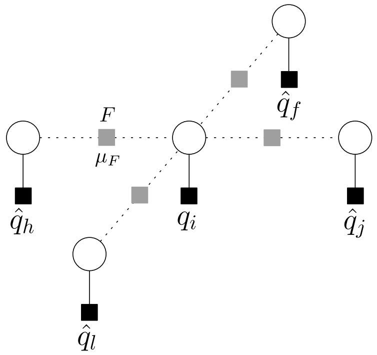


Fig. 3.13 Updating a single variable distribution q_i in naive mean field: the factor marginals μ_F are taken as product of variable marginals $\hat{q}_h$ and q_i. By fixing $\hat{q}_h$ and the other involved variable distributions, the remaining distribution q_i can be optimized over analytically.

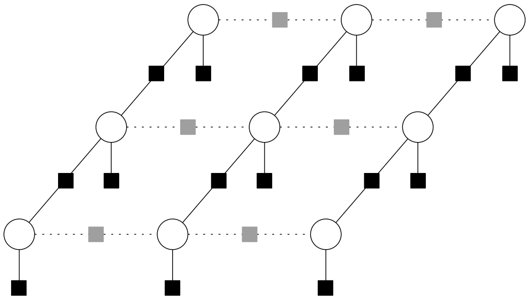


Fig. 3.14 Structured mean field approximation by taking larger tractable subgraphs of the factor graph. Here three chains are used and six factors are approximated using mean field. For each component the mean field update can be performed efficiently if inference for the component is tractable.

ascent. Because the objective (3.37) is differentiable the alternating optimization is known to converge to a locally optimal solution.

The mean field approach has a long history for inference in statistical models in general and in computer vision in particular. It has been used parameter estimation in MRF in the early 1990s, [48], and more recently to learn the parameters of CRF in image segmentation, [159], and stereo depth estimation, [161].

To improve the approximation of naive mean field we can take factorial distributions where each component is a larger subgraph of the original factor graph. This leads to the *structured mean field* approach [129]. An example is shown in Figure 3.14 where three chain-structured subgraphs are used and the shaded factors are removed from the approximating distribution. The resulting family $\mathcal{Q}$ of distributions is richer and therefore the approximation is improved. Optimizing (3.29) is still efficient but compared to the naive mean field approximation the entropies $H(q)$ now decompose over the subgraphs instead of individual variables. Each subgraph distribution can be updated by means of probabilistic inference on that subgraph. For details, see [160]. Further generalizations of the mean field approach are made in [170] and [167].

3.4 Sampling

Given a graphical model defining a probability distribution, we can use *sampling* methods to approximate expectations of functions under this distribution. That is, given $p(y|x,w)$ and an arbitrary function $h: \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ we can compute approximately the expected value of h,

$$\mathbb{E}_{y \sim p(y|x,w)}[h(x,y)]. \tag{3.41}$$

The ability to approximate (3.41) is important both for inference and parameter learning. For example, to perform probabilistic inference we define $h_{F,z_F}(x,y) = [\![y_F = z_F]\!]$ for all F and $z_F \in \mathcal{Y}_F$, obtaining marginal probabilities over factor states as

$$\mathbb{E}_{y \sim p(y|x,w)}[h_{F,z_F}(x,y)] = p(y_F = z_F | x, w). \tag{3.42}$$

For probabilistic parameter learning discussed in *Conditional Random Fields*, h is a *feature map* $\psi: \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$ and we evaluate the expectation of the feature map under the model distribution. While the computation of (3.41) subsumes the computation of marginal distributions and gradients of the log-partition function, it does not directly allow the computation of $\log Z(x,w)$, the log partition function itself.

3.4.1 Monte Carlo

The idea of evaluating (3.41) by sampling is to approximate the full expectation by means of a set of samples $y^{(1)}$, $y^{(2)}$, …, $y^{(S)}$ generated from $p(y|x,w)$. This is referred to as *Monte Carlo* approximation. We have, for a sufficiently large number S of samples that the sample mean approximates the true expectation,

$$\mathbb{E}_{y \sim p(y|x,w)}[h(x,y)] \approx \frac{1}{S} \sum_{s=1}^{S} h(x, y^{(s)}). \tag{3.43}$$

By the *law of large numbers* the approximation (3.43) converges arbitrarily close to the exact expectation if we use sufficiently many samples. For S independent samples the approximation error is then of the order $O(1/\sqrt{S})$, independent of the dimensionality of the problem. There are two problems, however. *The first problem* is that although we know the error decreases with more samples, for a given required approximation accuracy we do not know how many samples are sufficient. *The second problem* is that for a general graphical model obtaining exact samples $y^{(s)} \sim p(y|x,w)$ is a difficult problem itself.

Both problems can be mitigated by the use of the *Markov Chain Monte Carlo* (MCMC) method that we discuss now.

3.4.2 Markov Chain Monte Carlo

The basic idea of MCMC is to further approximate the approximation (3.43) by taking as $y^{(s)}$ not independent and identically distributed samples from $p(y|x,w)$, but instead to use a sequence $y^{(1)}, y^{(2)}, \ldots$ of *dependent samples* coming from a *Markov chain*. These samples can be produced by evaluating the unnormalized value of $p(y|x,w)$, i.e., we are no longer required to compute the log partition function. We first discuss briefly what a Markov chain is and then give two popular Markov chains for sampling from a graphical model.

Markov chains. Let us first briefly revisit the definition of a Markov chain as a *memoryless* random process whose future states depend only on the present state.

Definition 3.1 (Markov chain). Given a finite set $\mathcal{Y}$ and a matrix $P \in \mathbb{R}^{\mathcal{Y} \times \mathcal{Y}}$, then a random process $(Z_1, Z_2, \ldots)$ with Z_t taking values from $\mathcal{Y}$ is a *Markov chain with transition matrix* P, if

$$p(Z_{t+1} = y^{(j)} | Z_1 = y^{(1)}, Z_2 = y^{(2)}, \ldots, Z_t = y^{(t)}) \tag{3.44}$$

$$= p(Z_{t+1} = y^{(j)} | Z_t = y^{(t)}) \tag{3.45}$$

$$= P_{y^{(t)},y^{(j)}}. \tag{3.46}$$

Further details on Markov chains can be found in [55].

The above definition is for the case when $\mathcal{Y}$ is finite. An introduction to general Markov chains and their application to statistical inference is considered in [51, 126].

In the above definition the values of $P_{y^{(t)},y^{(j)}}$ are the *transition probabilities* of moving from state $y^{(t)}$ to state $y^{(j)}$. We can imagine the Markov chain as a walk on a directed graph that has as vertices all states $\mathcal{Y}$ and a directed edge $y^{(i)} \to y^{(j)}$ whenever $P_{y^{(i)},y^{(j)}} > 0$. When a Markov chain defined by P has two additional properties, *irreducibility* and *aperiodicity*, then it has a unique *stationary distribution*[3] $p(y)$ that satisfies $\sum_{y^{(i)} \in \mathcal{Y}} p(y^{(i)}) P_{y^{(i)},y^{(j)}} = p(y^{(j)})$ for all $y^{(j)} \in \mathcal{Y}$. The stationary distribution is the distribution we converge to when performing a random walk according to P.

The *key idea* of Markov chain Monte Carlo is to construct a Markov chain that has as its stationary distribution the true distribution of interest. By simulating the Markov chain for a number of time steps an approximate sample from the stationary distribution — the very distribution we are interested in — can be obtained. The *advantage* of doing so is that even in cases when it is hard to sample from the true distribution we can still efficiently simulate a Markov chain having this distribution as its stationary distribution.

By simulating the Markov chain long enough we obtain a single sample distributed approximately as $p(y|x,w)$. By repeating the procedure we could obtain an additional sample, but this would clearly be inefficient. Instead, in practise a single Markov chain is run for a large number of steps, taking a sequence of dependent samples $y^{(1)}, y^{(2)}, \ldots, y^{(t)}$ after each step, or every k steps, where k is usually small, say $k = 3$ or $k = 10$. Taking dependent samples from a Markov chain to evaluate (3.43) is still justified because of the *ergodic theorem* that guarantees that the influence of the initial and earlier iterate vanishes as we

[3] Also called equilibrium distribution of the Markov chain.

make keep making MCMC moves. A detailed discussion can be found in [125, Section 6.3.1].

Let us now discuss how a suitable Markov chain can be constructed. We discuss only the two most popular methods, the generally applicable *Metropolis-Hastings* method, and the special case of the *Gibbs sampler*.

Metropolis–Hastings chains. Originally proposed by Metropolis et al. [103] and extended by [57], the Metropolis–Hastings chain is widely applicable. Given the target distribution $\tilde{p}(y|x,w) \propto p(y|x,w)$ up to normalizing constants and given an additional *proposal distribution* $q(y'|y)$ the Markov chain is defined as shown in Algorithm 4.

Given a sample $y^{(t)}$ the algorithm iteratively samples from the proposal distribution $q(y'|y^{(t)})$ and either accepts or rejects the

Algorithm 4: Metropolis–Hastings Chain

```
MetropolisHastingsChain($\tilde{p}, q$)
Input:
    $\tilde{p}(y|x,w) \propto p(y|x,w)$, unnormalized target distribution
    $q(y'|y)$, proposal distribution
Output:
    $y^{(t)}$, sequence of samples with approximately $y^{(t)} \sim p(y|x,w)$
Algorithm:
$y^0 \leftarrow$ arbitrary in $\mathcal{Y}$
for $t = 1,\ldots,T$ do
    $y' \sim q(y'|y^{(t-1)})$ {Generate candidate}
    Compute acceptance probability
        $\sigma \leftarrow \min\left\{1, \frac{\tilde{p}(y'|x,w)q(y^{(t-1)}|y')}{\tilde{p}(y^{(t-1)}|x,w)q(y'|y^{(t-1)})}\right\}$   (3.47)
    Update
        $y^{(t)} \leftarrow \begin{cases} y' & \text{with probability } \sigma \text{ (accept),} \\ y^{(t-1)} & \text{otherwise (reject).} \end{cases}$   (3.48)
    output $y^{(t)}$
end for
```

candidate y'. In the computation only ratios of probabilities are used and hence the log partition function in $p(y|x,w)$ cancels out — equivalently, the unnormalized distribution $\tilde{p}(y|x,w)$ can be used.

The proposal distribution $q(y'|y^{(t)})$ can be constructed in a number of ways. A common method is to use a uniform distribution over a small set of variations of the current sample $y^{(t)}$, for example by allowing a single variable in $y^{(t)}$ to change its value. The Metropolis–Hastings method is so general that many variations have been proposed; for an in-depth review see [97].

Gibbs sampler. The Gibbs sampler, first proposed by Geman and Geman [49] is a special case of the Metropolis–Hastings chain in which each proposal is always accepted. The basic idea is that while sampling from $p(y|x,w)$ is hard, sampling from the conditional distributions $p(y_i|y_{V\setminus\{i\}},x,w)$ over small subsets of variables can be performed efficiently. By iteratively sampling variables from these distributions, conditioned on the state of the current sample, it is possible to show that the process defines a Markov chain with $p(y|x,w)$ as stationary distribution.

Algorithm 5: Gibbs Sampler

```
GibbsSampler($\tilde{p}$)
**Input:**
    $\tilde{p}(y|x,w) \propto p(y|x,w)$, unnormalized target distribution
**Output:**
    $y^t$, sequence of samples with approximately $y^{(t)} \sim p(y|x,w)$
**Algorithm:**
$y^{(0)} \leftarrow$ arbitrary in $\mathcal{Y}$
**for** $t = 1,\ldots,T$ **do**
    $y^{(t)} \leftarrow y^{(t-1)}$
    **for** $i \in V$ **do**
        Sample $y_i^{(t)} \sim p(y_i|y_{V\setminus\{i\}}^{(t)},x,w)$ using (3.49)
    **end for**
    **output** $y^{(t)}$
**end for**
```

In the algorithm, sampling from the conditional distribution is feasible because it only requires the unnormalized distribution $\tilde{p}$ and normalization over the domain of a single variable,

$$p(y_i|y^{(t)}_{V\setminus\{i\}},x,w) = \frac{p(y_i,y^{(t)}_{V\setminus\{i\}}|x,w)}{\sum_{y_i\in\mathcal{Y}_i} p(y_i,y^{(t)}_{V\setminus\{i\}}|x,w)} = \frac{\tilde{p}(y_i,y^{(t)}_{V\setminus\{i\}}|x,w)}{\sum_{y_i\in\mathcal{Y}_i} \tilde{p}(y_i,y^{(t)}_{V\setminus\{i\}}|x,w)}. \qquad (3.49)$$

One possible Gibbs sampler using a fixed order on the variables is shown in Algorithm 5. Performing the conditional sampling once on each variable is called a *sweep*, and this sampler outputs a new sample $y^{(t)}$ after each sweep. An application of this simple implementation is shown in Example 3.2. For discrete models the single site Gibbs sampler can be improved by minor changes to become the *metropolized Gibbs sampler* [97, Section 6.3.2], that is provably more efficient.

Example 3.2 (Gibbs sampling). We revisit the probabilistic inference problem of Example 2.1 where we visualized the posterior node marginals as shown in Figure 3.16 for the input image shown in Figure 3.15.

We run Algorithm 5 on the model distribution and visualize the convergence of the marginal foreground probability of the marked pixel in Figure 3.16. In particular, Figure 3.17 shows the cumulative mean as more and more samples are collected from the Gibbs sampler. The final probability is estimated to be 0.770. To get an idea of the behavior of this estimate, we repeat this experiment 50 times and visualize the different cumulative means in Figure 3.18. The estimated standard deviation is shown in Figure 3.19.

Examples of Monte Carlo sampling in computer vision. The Marr prize-winning work of Tu et al. [152] on hierarchical semantic image parsing makes use of sophisticated Monte Carlo techniques to jointly infer a hierarchical decomposition of a given image into components such as text, faces, and textures. Sampling based inference is

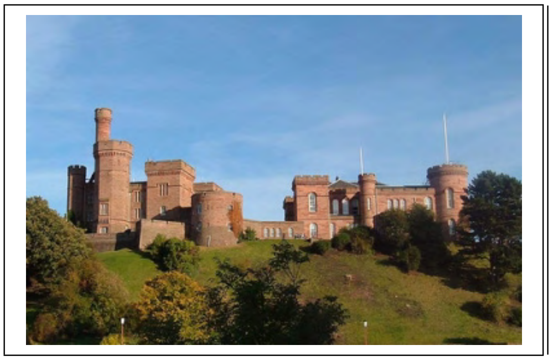


Fig. 3.15 Input image for man-made structure detection.

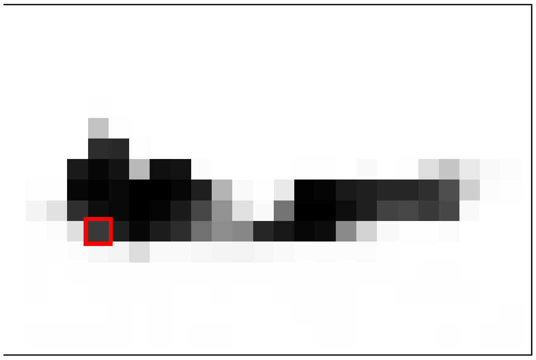


Fig. 3.16 Estimated posterior foreground marginals with marked pixel.

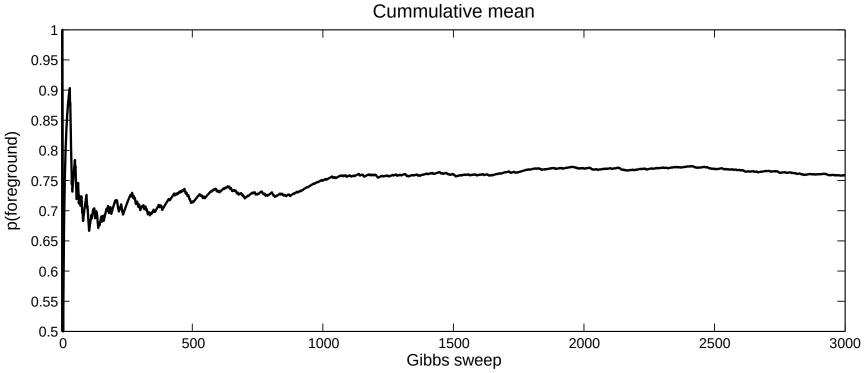


Fig. 3.17 Running mean of the estimated foreground probability of the marked pixel, produced by a simple Gibbs sampler running for 3,000 Gibbs sweeps after a burn-in phase of 100 sweeps.

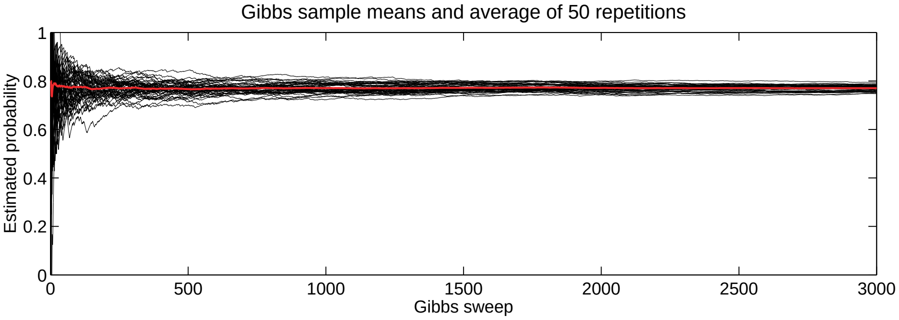


Fig. 3.18 A collection of 50 sample traces. The mean of the traces is shown in red. Using multiple sample traces allows an estimate of the sampler variance by the variance of the sample traces. After 3,000 sweeps the probability is estimated to be 0.770 with one unit standard deviation of 0.011.

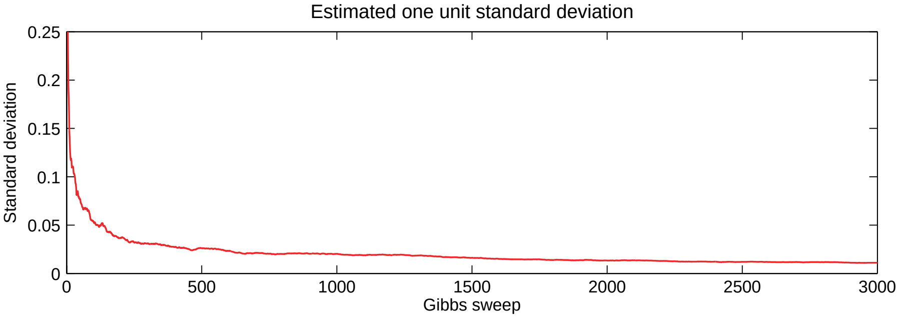


Fig. 3.19 Estimated standard deviation of the sampler result, in the order of $O(1/\sqrt{n})$ for n samples.

also popular in continuous random field models where exact inference is intractable. An example is the influential FRAME texture model of Zhu et al. [177] and continuous MRF natural image models [134]. An efficient sampler can also be used for approximate MAP inference by means of *simulated annealing*, a point we elaborate on in the next part.

Further reading. An introduction into MCMC methods for the use in machine learning is given in [108]. Markov chains and MCMC methods for discrete models such as the discrete factor graphs are discussed in the monograph [55]. A broader view on MCMC methods for general statistical models is provided in [126]. A general introduction into

Monte Carlo techniques and their wide applicability in science as well as guidelines on designing efficient samplers can be found in [97]. This reference also discusses importance sampling and sequential importance sampling, the other main branch of sampling methods that we have not described. Liang et al. [95] summarizes more recent advances, including adaptive MCMC algorithms.

4

Structured Prediction

4.1 Introduction

In the previous parts of this monograph we have discussed graphical models and probabilistic inference. While graphical models are flexible, we can generalize the way we make predictions using the setting of *structured prediction*.

In structured prediction we have a *prediction function* $f\colon \mathcal{X} \to \mathcal{Y}$ from an input domain $\mathcal{X}$ to a structured output domain $\mathcal{Y}$. The prediction function is defined in such a way that the actual prediction $f(x)$ for a given instance $x \in \mathcal{X}$ is obtained by maximizing an auxiliary evaluation function $g\colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ over all possible elements in $\mathcal{Y}$, such that

$$y^* = f(x) := \operatorname*{argmax}_{y \in \mathcal{Y}} \; g(x,y). \tag{4.1}$$

This is a generalization of the MAP inference task in graphical models. For example, when the model of interest is the probabilistic model $p(y|x)$, we can view the problem of finding the element $y^* \in \mathcal{Y}$ that maximizes $p(y|x)$ as an instance of (4.1) by defining $g(x,y) = p(y|x)$.

Another example of (4.1) is the linear model $g(x,y) = \langle w, \psi(x,y) \rangle$ with parameter vector w and feature map $\psi(x,y)$. Here, making a prediction corresponds to maximizing $\langle w, \psi(x,y) \rangle$ over y.

Solving the maximization problem (4.1) efficiently is the topic of this section. Because the pair $(g,\mathcal{Y})$ defines the problem completely, it is clear that all dependencies, constraints and relations of interest are encoded in g or $\mathcal{Y}$. As we will see, solving for y^* exactly often yields a hard optimization problem. Therefore, we will discuss is a set of general approaches to approximate y^* efficiently. A property shared by all these approaches is that they make explicit and exploit certain structure present in $\mathcal{Y}$ and g.

Although we highlight the successes of structured prediction methods within computer vision, at its core (4.1) is an optimization problem. As such, we will also make connections to the relevant literature from the optimization community.

4.2 Prediction Problem

We now formalize the above optimization problem and fix the notation used throughout this section.

Definition 4.1 (Optimization Problem). Given $(g,\mathcal{Y},\mathcal{G},x)$, with *feasible set* $\mathcal{Y} \subseteq \mathcal{G}$ over *decision domain* $\mathcal{G}$, and given an input instance $x \in \mathcal{X}$ and an *objective function* $g\colon \mathcal{X} \times \mathcal{G} \to \mathbb{R}$, find the optimal value

$$\alpha = \sup_{y \in \mathcal{Y}} g(x,y), \tag{4.2}$$

and, if the supremum exists, find an *optimal solution* $y^* \in \mathcal{Y}$ such that $g(x,y^*) = \alpha$.

We call $\mathcal{G}$ the *decision domain* and $\mathcal{Y}$ the *feasible set.* The reason for separating the two sets is for convenience: we will usually have a set $\mathcal{G}$ of simple structure — such as $\mathbb{R}^d$ — whereas $\mathcal{Y}$ will make explicit the problem specific structure. We say that an optimization problem is *feasible* if $\mathcal{Y}$ contains at least one element. In case we have $\mathcal{Y} = \mathcal{G}$, the problem is said to be *unconstrained.*

If $\mathcal{G}$ is a finite set, then the optimization problem is said to be a *discrete optimization problem.* Moreover, if $\mathcal{G}$ is the set of all subsets of some finite *ground set* Σ, that is, we have $\mathcal{G} = 2^{\Sigma}$, then the optimization problem is said to be a *combinatorial optimization problem.*

Clearly, if we are able to solve (4.2) we can evaluate $f(x)$, the prediction function of the model.

4.2.1 Parameterization and Feasible Sets

For a given structured prediction model there might exist different ways of how the evaluation function g and the decision domain $\mathcal{G}$ can be defined. To illustrate this point, consider the following classical example used in image denoising.

Example 4.1 (Ising Model and Markov Random Field). The Ising model is a popular physical model in statistical mechanics for modeling interactions between particles. It has been used to model image denoising tasks [49] and is the simplest case of a Markov random field. The basic model is as follows.

Given an undirected connected graph $G = (V, E)$, an interaction matrix $J \in \mathbb{R}^{V \times V}$ with $J = J^\top$, and a vector $h \in \mathbb{R}^V$, the task is to recover for each $i \in V$ a binary state agreeing with the sign of h_i as well as the pairwise interaction terms.

In the original Ising model there are no observations $x \in \mathcal{X}$.[1] Two possible formulations as structured prediction problem are given with the following choices for $\mathcal{G}$, $\mathcal{Y}$, and g.

(1) Ising model with external field [49].

$$\mathcal{Y} = \mathcal{G} = \{-1, +1\}^V, \tag{4.3}$$

$$g(y) = \frac{1}{2} \sum_{(i,j) \in E} J_{i,j} y_i y_j + \sum_{i \in V} h_i y_i, \tag{4.4}$$

(2) Markov random field parameterization [160].

$$\mathcal{G} = \{0, 1\}^{(V \times \{-1,+1\}) \cup (E \times \{-1,+1\} \times \{-1,+1\})}, \tag{4.5}$$

[1] We could introduce observational data into the model by making J and h functions of an observation, i.e., $J\colon \mathcal{X} \to \mathbb{R}^{V \times V}$ and $h\colon \mathcal{X} \to \mathbb{R}^V$, such that in (4.4) and (4.5) we replace J and h with $J(x)$ and $h(x)$, respectively.

$$\begin{aligned}
\mathcal{Y} = \{y \in \mathcal{G}\colon & \forall i \in V : y_{i,-1} + y_{i,+1} = 1, \\
& \forall (i,j) \in E\colon y_{i,j,+1,+1} + y_{i,j,+1,-1} = y_{i,+1}, \\
& \forall (i,j) \in E\colon y_{i,j,-1,+1} + y_{i,j,-1,-1} = y_{i,-1}\}, \qquad (4.6)
\end{aligned}$$

$$\begin{aligned}
g(y) = & \frac{1}{2} \sum_{(i,j) \in E} J_{i,j}(y_{i,j,+1,+1} + y_{i,j,-1,-1}) \\
& - \frac{1}{2} \sum_{(i,j) \in E} J_{i,j}(y_{i,j,+1,-1} + y_{i,j,-1,+1}) \\
& + \sum_{i \in V} h_i(y_{i,+1} - y_{i,-1}). \qquad (4.7)
\end{aligned}$$

Both formulations have the same optimal value and the same number of elements in their feasible sets. While (1) is an unconstrained problem, its objective function is a quadratic function in the variables y. In contrast, the parameterization chosen in (2) leads to an objective function that is linear in y, at the cost of a more complicated feasible set described by a collection of linear equality constraints.

The choice of parameterization is an important modeling decision that affects the solution algorithm used to solve the prediction problem. As an example, in the above formulations, the prediction problem resulting from the second formulation is an integer linear programming problem that in turn lends itself to relaxation approaches. In contrast, the first formulation could be used in a local search approach as it is an unconstrained problem and its objective function can be evaluated efficiently.

4.2.2 Objective Function

The objective function $g(x,y)$ encodes everything relevant to judge the quality of a solution y. In computer vision applications it is often the case that a good prediction y shall satisfy multiple goals. For example, if y corresponds to a segmentation of an image we would like y to be aligned with image edges. But also we prefer solutions that are smooth as measured for example by the mean curvature. Two or more objectives might be contradicting each other and in order to perform optimization as discussed in this section we need to use a single objective function.[2]

In general we often face the situation that we have multiple objective functions $g_1 \colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}, \dots, g_K \colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ and we define $g \colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ as a linear combination

$$g(x,y) = \sum_{k=1}^{K} \lambda_k g_k(x,y), \tag{4.8}$$

where $\lambda_k \geq 0$ are weights that determine the relative importance of $g_1, \dots, g_K$. In practice the values of λ are often set manually. Another popular method is to tune them on a hold-out set or by cross validation.

Another consideration when building an objective function is computational complexity. During optimization the objective function is evaluated repeatedly and therefore it should in general be efficient to evaluate.

4.3 Solving the Prediction Problem

For many computer vision models, the prediction problem (4.1) is hard to solve. Even conceptually simple models such as grid-structured binary state random fields give rise to NP-hard prediction problems. For models that have prediction problems that are solvable in polynomial time a naive enumeration approach might be computationally infeasible and more efficient alternatives need to be developed to solve instances of practical relevance.

For many hard prediction problem arising in computer vision there has been considerable research effort in devising efficient solution algorithms to obtain approximately optimal solutions. In all cases, the hardness of the original problem remains but by *giving up* other algorithmic properties a practical algorithm can be obtained.

We divide the existing approaches to solve the prediction problem by *what* is being given up: *generality*, *optimality*, *worst-case complexity*, *integrality* and *determinism*. Guarantees on all these properties are

[2] The field of *multi-objective optimization* deals with finding solutions of multiple objective functions. The produced solutions satisfy a generalized notion of optimality.

desirable but for hard prediction problems cannot be achieved at the same time.

Giving up *generality* refers to identifying from a given problem class a subset of sufficiently rich problems that can be solved efficiently. The hypothesis class is then restricted to only these prediction functions.

Giving up *optimality* means that the output of a solution algorithm is no longer guaranteed to be optimal. Many iterative solution approaches such as local search fall in this category.

Giving up *worst-case complexity* describes algorithms which might be efficient and optimal for practical problem instances but lack a guaranteed worst-case complexity that is polynomial in the problem size. Implicit enumeration methods such as branch-and-bound search belong to this class of methods.

Giving up *integrality* subsumes all approaches in which the feasible set is enlarged in order to simplify the prediction problem, thereby a *relaxation* to the original problem is obtained. We will discuss the most common techniques to construct such relaxations, based on integer linear programming and mathematical programming decomposition approaches.

By giving up *determinism* we mean algorithms that use randomness in order to obtain solutions that are optimal most of the times. Classic techniques such as simulated annealing on distributions simulated by Markov chains and recent *randomized algorithms* belong to this category.

The above classification of algorithms is helpful in identifying approaches for a novel prediction problem. However, it is not a strict classification and many methods used in computer vision fall in two or more categories. We now discuss the five categories in detail and give successful examples from computer vision for each.

4.4 Giving up Generality

Prediction problems that are hard in the general case often have special cases that are tractable. For example, while MAP inference in general graphical models is hard, we already mentioned the special case of tree-structured graphical models that are solvable exactly. In this case the special structural restriction makes the problem tractable. In contrast, for some problems it is not the structure in the model but in the coefficients that defines a tractable subclass.

One popular example in this category are binary random fields, that are in general hard to optimize exactly. But for the special case in which the pairwise interactions are restricted to be *regular*, the problem becomes efficiently solvable. The most efficient way to solve these instances is by means of graph cuts, as we explain now.

4.4.1 Binary Graph Cuts

The binary graph cut method is able to globally minimize a restricted class of energy functions on binary variables. To do this, an undirected auxiliary graph is constructed from the energy function. This graph contains two special nodes, the *source* s and the *sink* t, as well as a non-negative weight for each edge. By solving for the *minimum* s–t *cut* of the graph, that is, the cut separating the nodes s and t that has the smallest overall weight, the optimal solution to the original energy minimization problem can be obtained. Because the minimum s–t cut problem can be solved very efficiently, the overall method scales to millions of variables.

The class of binary energy functions amenable to graph cuts is of the form

$$E(y;x,w) = \sum_{F \in \mathcal{F}_1} E_F(y_F;x,w_{t_F}) + \sum_{F \in \mathcal{F}_2} E_F(y_F;x,w_{t_F}), \tag{4.9}$$

where $\mathcal{F}_1$ and $\mathcal{F}_2$ denote unary and pairwise factors, respectively. The unary energies are restricted so as to satisfy

$$E_F(y_i;x,w_{t_F}) \geq 0, \tag{4.10}$$

and likewise for the pairwise energies it must hold that

$$\begin{aligned} E_F(y_i,y_j;x,w_{t_F}) &= 0, & \text{if } y_i = y_j, \quad (4.11)\\ E_F(y_i,y_j;x,w_{t_F}) &= E_F(y_j,y_i;x,w_{t_F}) \geq 0, & \text{otherwise.} \quad (4.12) \end{aligned}$$

Therefore, the pairwise factors encourage their adjacent variables to take the same state.

Given these assumptions, we construct the auxiliary graph as shown in Figure 4.1. Each binary variable becomes one node in the graph, but additionally we add a source node s and a sink node t. The variable nodes are connected with pairwise undirected edges whenever there is a pairwise factor in the factor graph. Additionally each variable node is connected to both the source and sink nodes. To fully specify the s–t min cut problem, we have to specify the edge weights for each edge added. These are as shown in Table 4.1.

After solving the graph cut problem, we can reconstruct the minimizing state of the energy (4.9) by determining which side of the cut the variable lies on. Variable connected to the source s take the state 0,

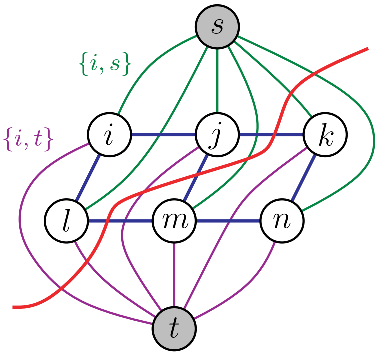


Fig. 4.1 Undirected graph cut construction of [29]. A *source* node (s) and a *sink* node (t) are connected to all variable nodes. The undirected s–t cut of minimum weight (shown in red) minimizes the energy (4.9). The optimal solution $y_i = y_j = y_l = 0$, $y_k = y_m = y_n = 1$ is determined by the side of the cut each variable lies.

Table 4.1. Edge weights derived from the energy function (4.9). The graph construction remains valid if negative energies are used in pairwise or unary factors, but in this case the minimum s–t cut problem becomes NP-hard in general.

Edge	Graph cut weight
$\{i,j\}$	$E_F(y_i = 0, y_j = 1; x, w_{t_F})$
$\{i,s\}$	$E_F(y_i = 1; x, w_{t_F})$
$\{i,t\}$	$E_F(y_i = 0; x, w_{t_F})$

and variables connected to the sink t have state 1. In case the solution to (4.9) is not unique, we obtain only one of the minimizers.

The assumptions made on the energy function, (4.10) for the unary energy terms, and (4.11)–(4.12) for the pairwise terms may appear quite strict at first, but in fact a larger class of energy functions can be equivalently transformed such as to satisfy them. For example, the unary condition $E_F(y_i;x,w) \geq 0$ can always be ensured by adding a constant to both unary energies, that is, setting $E'_F(y_i = 0;x,w) = E_F(y_i = 0;x,w) + C$ and $E'_F(y_i = 1;x,w) = E_F(y_i = 1;x,w) + C$, with a sufficiently large constant $C \geq 0$. Because this addition is made to both states, only the value of the energy (4.9) is changed, but not the labeling minimizing it. Similar transformations can be made for the pairwise energies, and a complete characterization of graph cut solvable binary energy functions has been given by Kolmogorov and Zabin [80] and Freedman and Drineas [44]. Their main results characterize general energy functions involving interactions between two and three variables with binary states by stating sufficient conditions such that a graph cut problem can be constructed that has as minimizer the minimizer of the energy function. In particular, for energy functions with only unary and pairwise terms, Kolmogorov and Zabin [80] show the following theorem.

Theorem 4.1 (Regular Binary Energies). Let

$$E(y;x,w) = \sum_{F \in \mathcal{F}_1} E_F(y_F;x,w_{t_F}) + \sum_{F \in \mathcal{F}_2} E_F(y_F;x,w_{t_F}), \qquad (4.13)$$

be a energy function of binary variables containing only unary and pairwise factors. The discrete energy minimization problem $\operatorname{argmin}_y E(y;x,w)$ is representable as a graph cut problem if and only if all pairwise energy functions E_F for $F \in \mathcal{F}_2$ with $F = \{i,j\}$ satisfy

$$E_{i,j}(0,0) + E_{i,j}(1,1) \leq E_{i,j}(0,1) + E_{i,j}(1,0). \qquad (4.14)$$

Such energies are called *regular*.

The conditions (4.14) can be understood as requiring that adjacent nodes must have a lower energy if they are labeled with the same state than when they have different states, a property often referred to as "associative" or "attractive" potential. For interactions involving three nodes, this holds if each *projection* onto two variables satisfies the above condition. Details are given by Kolmogorov and Zabin [80].

Once we have constructed the graph for an energy function, there exist multiple algorithms to solve the corresponding min cut problem. They have been reviewed in Boykov and Kolmogorov [26]. Most implementations rely on the fact that instead of the *min cut* one can solve an equivalent *max flow* problem, which follows from linear programming duality [116].

Example 4.2 (Figure-ground image segmentation). *Foreground–background*, or *figure-ground*, image segmentation can naturally be formulated as MAP prediction of a binary energy function. We take $\mathcal{X}$ as the set of natural images and $\mathcal{Y}$ as the set of all possible binary pixel labelings for the image where a predicted value 1 for a pixel indicates *foreground* and 0 indicates *background.* From a suitably parametrized probability distribution $p(y|x)$, we obtain a prediction function

$$g(x,y,w) = \sum_{i\in V} \log p(y_i|x_i) + w \sum_{(i,j)\in E} C(x_i,x_j)[\![y_i \neq y_j]\!], \qquad (4.15)$$

where V is the set of pixels, E is the set of neighboring pixel pairs, $w \geq 0$ is a scalar parameter, and $p(y_i|x_i)$ is estimated using a color model of the expected foreground class. The term $C(x_i,x_j) \geq 0$ is a fixed penalty function evaluating the intensity or color contrast between two adjacent pixels i and j. A common choice is $C(x_i,x_j) = \exp(\gamma\|x_i - x_j\|^2)$, where γ is estimated from the mean edge strength in the image, as suggested by Blake et al. [20].

With this choice, the function (4.15) has only unary and pairwise interactions between binary variables and the pairwise terms fulfill the regularity conditions of Theorem 4.1. Consequently, we can find the global maximizer of (4.15) by the *graph cuts* method.

Maximizing (4.15) strives to label each pixel with the preference of the color model but also to not change the label unless there is an edge in the image. Figures 4.2 and 4.3 show an example. By varying the strength of the penalization one obtaines segmentations that are

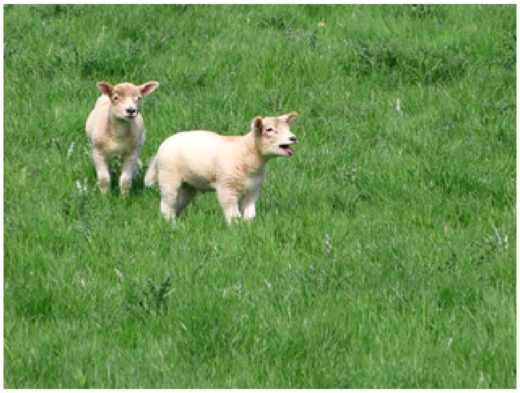


Fig. 4.2 A natural image to be segmented. (Image source: `http://pdphoto.org`)

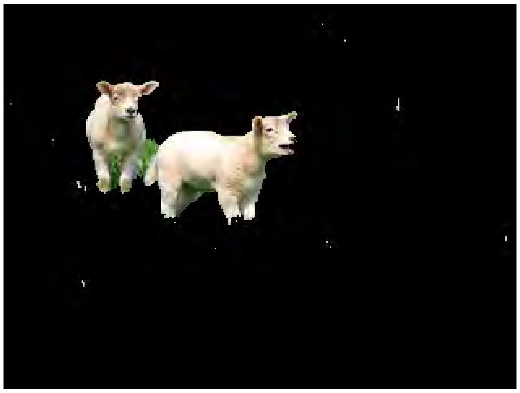


Fig. 4.3 Resulting foreground region.

smoother, but less detailed (Figure 4.4), or more detailed, but noisier (Figure 4.5).

The identification of a tractable subclass of instances can be hard in general. If the problem has a natural interpretation in terms of a graph then structural properties of the graph could lead to tractable classes. One example is in the context of MAP inference in graphical models, where it is known that the MAP inference problem can be solved with a complexity that is exponential in the so called *treewidth* of the graph. Because tree-structured models have a treewidth of 1

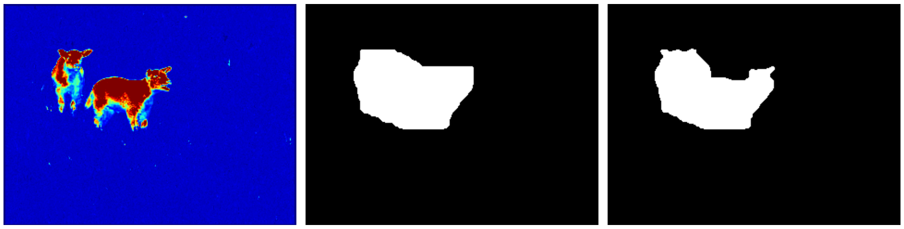


Fig. 4.4 Left: heatmap of unary potential values. Right: segmentation masks for large w.

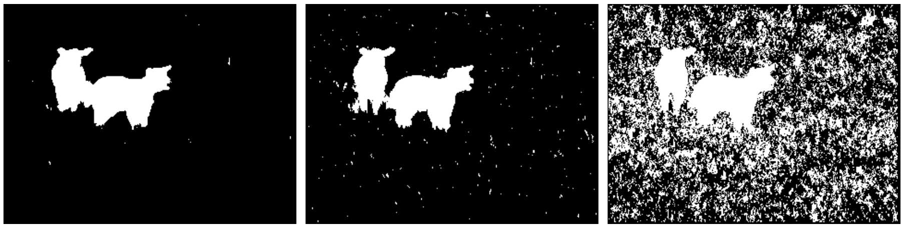


Fig. 4.5 Segmentation masks for medium and small w.

these can be solved efficiently, but low treewidth graphs also remain tractable.

Another example is graph *planarity*.[3] If a graph is planar many combinatorial optimization problems on the graph turn out to be tractable. An efficient algorithm for a restricted class of binary planar MRFs has been proposed by Schraudolph and Kamenetsky [136]. Outerplanar MRFs have been used by Batra et al. [11] to approximately solve hard random field instances. Even in the case of polynomial-time solvable instances the computational complexity can be reduced by exploiting planarity, as shown by Schmidt et al. [132, 133]. Note that planarity is distinct from treewidth in that a planar graph may have high treewidth. For instance, an $n \times n$ grid graph has treewidth n but is clearly planar.

Example 4.3 (Class-independent Object Hypotheses). The computational complexity of object recognition in natural images can often be reduced significantly, if one has a set of *object hypothesis* regions available. Assuming that at least one of these will coincide well with the object, learning and prediction need only consider a

[3] A graph is planar if it can be embedded in the plane without any crossing edges.

relatively small number of reasonable candidate regions instead of having to search the much larger set of all possible locations.

Carreira and Sminchisescu [30] recently proposed the *constrained parametric min cuts (CPMC)* method for generating class-independent object hypotheses based on a collection of figure-ground segmentation tasks. They define an energy function with contrast dependent pairwise factors, $E_{ij} = wC(x_i, x_j)[\![y_i \neq y_j]\!]$, for neighboring pixels i and j, thereby making use of the class-independent assumption that object boundaries should coincide with image edges. It is less clear how one can construct unary factors without making *a priori* assumptions about object appearance. CPMC's solution to this problem consists of trying many different choices, each one giving rise to one object hypothesis. It defines a set of *seed* points by laying a coarse grid over the image and iteratively picking the center points of the grid cells. For each seed, it defines a unary factor that is strong enough to force the corresponding pixel to lie in the foreground. Unary factors of opposite sign are used to force some or all image boundaries to be labeled as background. For all remaining pixels the unary factors are set identically as a variable value that encodes a bias in favor of more or fewer pixels to be labeled as foreground. Each parameter choice results in a submodular energy function that is minimized by the graph cuts algorithm.

Even though CPMC requires many segmentation to be computed it remains computationally tractable, because one only has to loop over the seeds, whereas all minimizing solution for different values of the unary factors can be found by a single call to a *parametric min cut* solver [117].

Figure 4.6 shows examples of the resulting object hypotheses for the image in Figure 4.2. As one can see, several object hypothesis do in fact overlap strongly with the correct object locations.

4.5 Giving up Optimality

Solving the prediction problem optimally is hard but in many applications it is not necessary to obtain *the* optimal solution and instead any

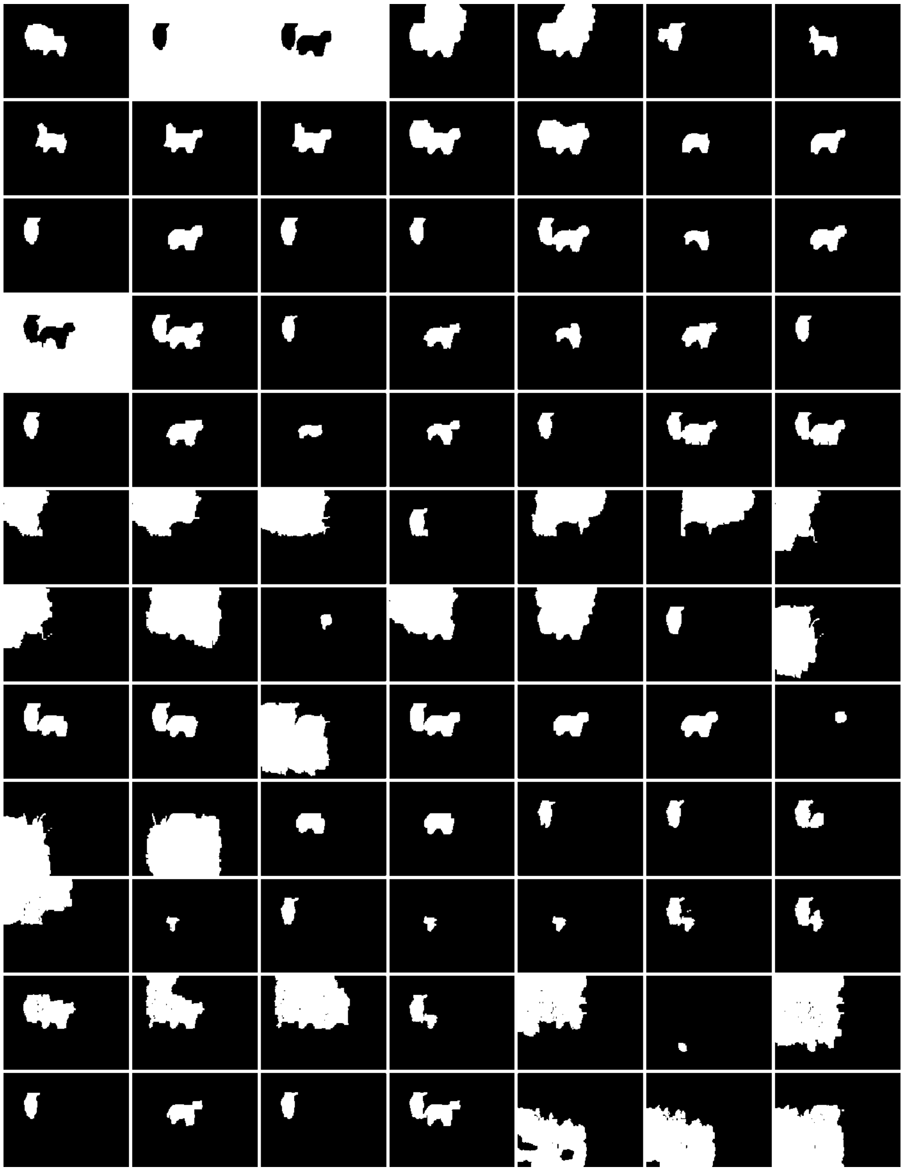


Fig. 4.6 CPMC object proposals for the image in Figure 4.2. Despite the method not knowing what kind of objects are visible in theimage, some of the segmentations coincide well with how a human would partition the image into an *object* and a *nonobject* regions.

close-to-optimal solution is sufficient. The set of good but suboptimal solutions might be large and finding one element from it could be easy compared to identifying the optimal solution.

Another reason why we might be satisfied with a good suboptimal solution is *model uncertainty* resulting from both using a simplified or wrong model and from the parameter estimation error [25]. Parameters of the prediction function are estimated from a finite and usually small set of training data, leading to an *estimation error*. If $y^* \in \mathcal{Y}$ is the optimal prediction for a learned prediction function, then the preference for y^* might largely be due to this estimation error. If we consider the set of prediction functions "nearby" the learned predictor — that is, within range of the estimation error — then all prediction functions within this set are reasonable, yet they produce different predictions. The set of optimal solutions from all these prediction functions contains not only y^* but many more solutions, all considered suboptimal by the original prediction function but by themselves being reasonable predictions. Typically the learned prediction function will assign these solutions a value close to the optimal value.

Local search methods are a popular class of algorithms in which optimality of the solution cannot be guaranteed. In a local search method an initial feasible solution is improved iteratively by finding improved solutions within a *neighborhood* of the current solution. The method terminates when no further improvement can be made. For many structured prediction problems a simple and natural local search algorithm can be formulated and we will discuss the general scheme in detail.

Giving up optimality does not imply that the methods used come without theoretical guarantees. *Approximation algorithms* [155] offer worst-case *a priori* guarantees on the value of the returned approximate solution with respect to the value of the true but unknown optimal solution. For maximization problems such as (4.2) the guarantee takes the form of a *relative performance guarantee* by means of a scalar factor $0 < \rho < 1$ such that when the approximation algorithm returns a solution y, it holds that $\rho g(x,y^*) \leq g(x,y) \leq g(x,y^*)$.

In practice suboptimal prediction methods such as local search are also used during training of a structured prediction system. Although this often works well empirically, the interaction between the suboptimal solutions provided and the training method are not well understood. In fact, recent results suggest that the use of suboptimal predictions during training can lead to prediction functions with bad performance [39, 86, 99], whereas alternative approaches based on relaxing the prediction problem work well in training. Nevertheless, approaches such as local search are among most popular approximations for test-time prediction.

4.5.1 Local Search

Local search is an umbrella term for optimization methods that iteratively improve a feasible solution by optimizing with respect to subsets of the feasible set. Even in case optimization over the original feasible set is hard, by restricting the optimization problem to a much smaller set the resulting *local optimization problem* can be solved efficiently. Typically these smaller subsets are *neighborhoods* around the currently best known feasible solution. This idea is illustrated in Figure 4.7.

A generic local search algorithm is given in Algorithm 6. The algorithm takes as input an initial feasible solution and a set of neighborhood relations. In general, a neighborhood relation does not need to be fixed but can vary over time. In Algorithm 6 the neighborhood is given as $\mathcal{N}_s : \mathcal{Y} \to 2^{\mathcal{Y}}$, where the index $s = 1, \dots, S$ indexes a set of relations. In each iteration t, the algorithm attempts to improve the currently best known solution y^t by solving the restricted optimization problem

$$\operatorname*{argmax}_{y \in \mathcal{N}_s(y^{t+1})} g(x,y). \tag{4.16}$$

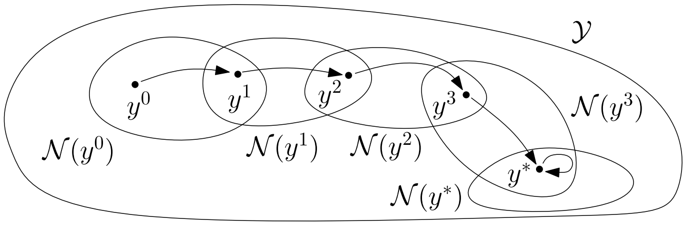


Fig. 4.7 Illustration of local search: an initial solution y^0 is improved by finding the best solution within a neighborhood $\mathcal{N}(y^0)$, obtaining a new solution $y^1 \in \mathcal{N}(y^0)$. The process is continued until a solution y^* is reached such that no further improvement within its neighborhood can be made.

Algorithm 6: Local Search

```
$y^* = \textsc{LocalSearch}(x, y^0, \mathcal{N}_s, S)$
**Input:**
    $x \in \mathcal{X}$ instance information
    $y^0 \in \mathcal{Y}$ initial solution
    $\mathcal{N}_s \colon \mathcal{Y} \to 2^{\mathcal{Y}}$ neighborhood mapping at step $s$
    $S \in \mathbb{N}$, $S \geq 1$ number of neighborhoods in each cycle
**Output:**
    $y^* \in \mathcal{Y}$ local optimal solution in $\mathcal{N}_s(y^*)$ for all $s = 1, \ldots, S$
**Algorithm:**
$t \leftarrow 0$
**for** $t = 0, 1, \ldots$ **do**
    $y^{t+1} \leftarrow y^t$
    **for** $s = 1, \ldots, S$ **do**
        $y^{t+1} \leftarrow \operatorname{argmax}_{y \in \mathcal{N}_s(y^{t+1})} g(x, y)$   {Maximize within
        neighborhood}
    **end for**
    **if** $y^{t+1} = y^t$ **then**
        **break** {Local optima w.r.t. $\mathcal{N}_s(y^t)$ for all $s = 1, \ldots, S$}
    **end if**
**end for**
$y^* \leftarrow y^t$
```

Because each neighborhood is assumed to contain the current solution, i.e., $y^t \in \mathcal{N}_s(y^t)$, the algorithm will never decrease the objective function, and therefore produces a sequence of monotonically improving solutions. A solution y^* is locally optimal if it can no longer be improved with respect to any neighborhoods $\mathcal{N}_s(y^*)$. The neighborhoods are typically chosen such that the restricted problem becomes tractable. Let us illustrate this point by a classic example, the iterated conditional modes (ICM) algorithm.

Example 4.4 (Iterated Conditional Modes). The *iterated conditional modes (ICM)* algorithm is a local search method that can be applied to any given discrete factor graph but has been originally proposed for images by Besag [17]. For the task of image denoising, the factor graph often has a structure as shown in Figure 4.8, where the dependent variables are located on the 2D pixel grid lattice.

If all but one variable were observed, then solving for the MAP state of the single dependent variable would be easy. Such situation is shown in Figure 4.9. The *iterated conditional modes* (ICM) method uses this property to iteratively update one variable at a time, keeping all other variables fixed. This is a local search method with neighborhood relation

$$\mathcal{N}_s(y) = \{(y_1,\ldots,y_{s-1},z_s,y_{s+1},\ldots,y_S)|z_s \in \mathcal{Y}_s\}, \tag{4.17}$$

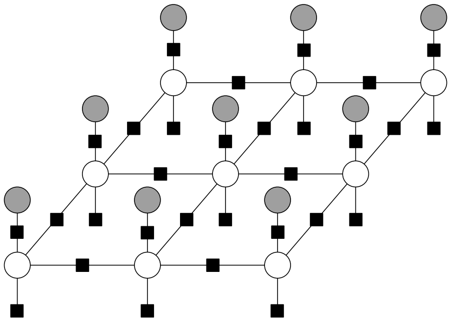


Fig. 4.8 Factor graph for a simple random field model with observation variables (shaded gray) and dependent variables (white) to be inferred.

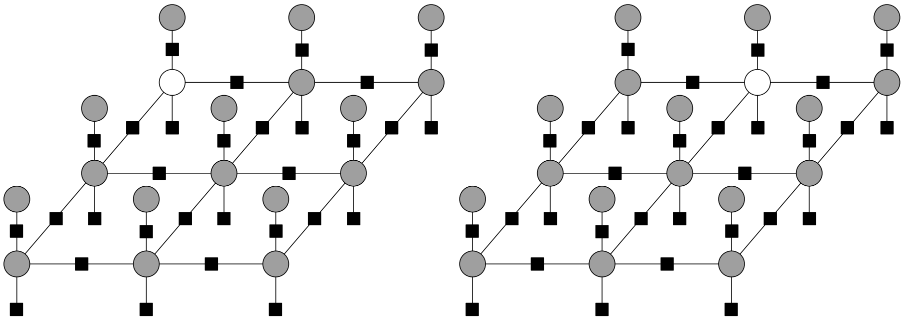


Fig. 4.9 ICM update for one variable as local search: keeping all dependent variables but the first fixed effectively treats them as observed. The single remaining variable can be optimized efficiently. The figure on the right shows an ICM update at a different site.

where $s = 1,\dots,S$ and $S = |V|$ indexes the dependent variables of the model. When we iterate over all the neighborhoods in Algorithm 6, we effectively optimize the values of variables y_s one-by-one, improving the overall objective function.

The solution returned by the local search method guarantees local optimality with respect to the neighborhoods. Clearly, the larger the neighborhood, the stronger this local optimality guarantee becomes. The largest possible neighborhood — the original set $\mathcal{Y}$ itself — recovers the original problem. In general, we would like to select the largest neighborhood relation that still allows for efficient optimization. Successful local search approaches use neighborhoods that exploits problem structure.

In the ICM example local search becomes a coordinate descent method where in each iteration a subset of the optimization variables are fixed and optimization is restricted to the remaining variables. This is a common way to construct search neighborhoods and in this sense local search generalizes coordinate descent methods. There are at least two other common names for local search methods, *move-making* methods [73], and *very large-scale neighborhood search* [2].

We first discuss a generalization of the ICM method, the *block ICM* method, before we explain the most popular local search method in computer vision, the class of multilabel *graph cut methods*.

Example 4.5 (Block Iterated Conditional Modes). The previous ICM method selects one variable at a time and optimizes within the neighborhood by explicitly evaluating the energy for all values this variables takes. In the block ICM method the neighborhood is enlarged by allowing a larger subset of variables to change [68, 72].

Fixing a subset of variables to their current values corresponds to *conditioning* the probability distribution. Optimization within the neighborhood defined by the resulting conditioned distribution remains efficiently solvable as long as the subset of variables form a tree-structured subgraph in the original factor graph. Then, the max-product belief propagation algorithm can be used to solve (4.16).

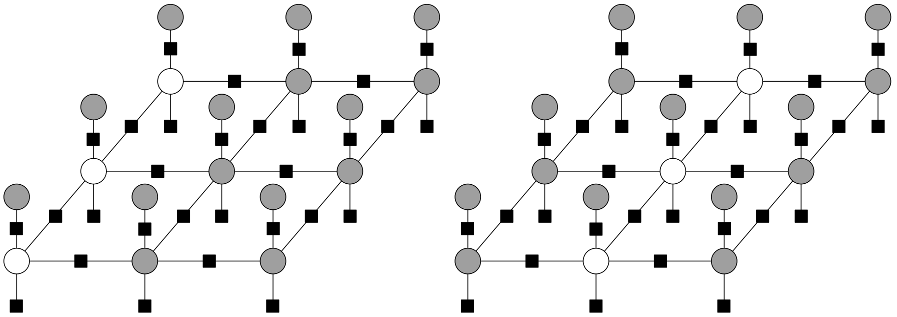


Fig. 4.10 Tractable chain-structured subgraphs used in block ICM: optimization over tree-structured subgraphs remains efficiently solvable. The figure on the right shows another chain-structured subgraph inducing a neighborhood with size exponential in the number of variables, i.e., $|\mathcal{Y}_i \times \mathcal{Y}_j \times \mathcal{Y}_k|$.

Whereas the original ICM method neighborhood is as large as the number of labels the variable can take, the search space used by the block ICM neighborhoods is exponential in the number of variables optimized over. For grid-structured graphs a typical subset of variables induced by chains is shown in Figure 4.10.

4.5.2 Graph Cuts

The most popular local search method in computer vision is the α-expansion *graphcut method* for multi-label discrete MAP inference problems [28]. A decade since their introduction they remain popular because they are both efficient and provide high-quality solutions.

Boykov et al. [28] proposed two neighborhoods, the "α-expansion" neighborhood $\mathcal{N}_\alpha$: $\mathcal{Y} \times \mathbb{N} \to 2^{\mathcal{Y}}$ and the "α–β-swap" neighborhood $\mathcal{N}_{\alpha,\beta}$: $\mathcal{Y} \times \mathbb{N} \times \mathbb{N} \to 2^{\mathcal{Y}}$. We will discuss both neighborhoods separately, starting with the simpler α–β-swap. Let us first make some assumptions. We are interested in maximizing $g(x,y) = -E(y;x)$, where the energy function

$$E(y;x) = \sum_{i \in V} E_i(y_i;x) + \sum_{(i,j) \in \mathcal{E}} E_{i,j}(y_i,y_j;x)$$

decomposes into unary and pairwise terms. We require for both the α-expansion and the α–β-swap neighborhoods that the pairwise energy terms are a *semi-metric*, satisfying for all $(i,j) \in \mathcal{E}$, $(y_i,y_j) \in \mathcal{Y}_i \times \mathcal{Y}_j$ the conditions

$$E_{i,j}(y_i,y_j;x) = 0 \Leftrightarrow y_i = y_j, \tag{4.18}$$

$$E_{i,j}(y_i,y_j;x) = E_{i,j}(y_j,y_i;x) \geq 0. \tag{4.19}$$

The first condition (4.18) is the *identity of indiscernibles*, the second condition (4.19) is *symmetry*. Moreover, the α-expansion additionally requires the pairwise energies to be a true metric, i.e., to satisfy (4.18), (4.19) and for all $(i,j) \in \mathcal{E}$, for all $(y_i,y_j) \in \mathcal{Y}_i \times \mathcal{Y}_j$, for all $y_k \in \mathcal{Y}_i \cap \mathcal{Y}_j$ that

$$E_{i,j}(y_i,y_j;x) \leq E_{i,j}(y_i,y_k;x) + E_{i,j}(y_k,y_j;x), \tag{4.20}$$

which is the well known *triangle inequality*. We are now ready to consider the definition of the neighborhoods.

4.5.2.1 $\boldsymbol{\alpha}$–$\boldsymbol{\beta}$-swap

The α–β-swap neighborhood is defined as follows.

$$\begin{aligned} \mathcal{N}_{\alpha,\beta} &: \mathcal{Y} \times \mathbb{N} \times \mathbb{N} \to 2^{\mathcal{Y}}, \\ \mathcal{N}_{\alpha,\beta}(y,\alpha,\beta) &:= \{z \in \mathcal{Y} : z_i = y_i \text{ if } y_i \notin \{\alpha,\beta\}, \\ &\qquad\qquad \text{otherwise } z_i \in \{\alpha,\beta\}\}. \end{aligned} \tag{4.21}$$

Therefore the neighborhood $\mathcal{N}_{\alpha,\beta}(y,\alpha,\beta)$ contains the solution y itself as well as all variants in which the nodes labeled α or β are free to change their label to either β or α, respectively. Finding the minimizer becomes a binary labeling problem because the only two states of interest are α and β. We can decompose the following minimization problem.

$$\begin{aligned} y^{t+1} &= \operatorname*{argmin}_{y \in \mathcal{N}_{\alpha,\beta}(y^t,\alpha,\beta)} E(y;x) \\ &= \operatorname*{argmin}_{y \in \mathcal{N}_{\alpha,\beta}(y^t,\alpha,\beta)} \sum_{i \in V} E_i(y_i;x) + \sum_{(i,j) \in E} E_{i,j}(y_i,y_j;x) \end{aligned}$$

$$
\begin{aligned}
= \operatorname*{argmin}_{y\in\mathcal{N}_{\alpha,\beta}(y^t,\alpha,\beta)} \Bigg[& \underbrace{\sum_{\substack{i\in V,\\ y_i^t\notin\{\alpha,\beta\}}} E_i(y_i^t;x)}_{\text{constant}} + \underbrace{\sum_{\substack{i\in V,\\ y_i^t\in\{\alpha,\beta\}}} E_i(y_i;x)}_{\text{unary}} \\
+ & \underbrace{\sum_{\substack{(i,j)\in E,\\ y_i^t\notin\{\alpha,\beta\},y_j^t\notin\{\alpha,\beta\}}} E_{i,j}(y_i^t,y_j^t;x)}_{\text{constant}} + \underbrace{\sum_{\substack{(i,j)\in E,\\ y_i^t\in\{\alpha,\beta\},y_j^t\notin\{\alpha,\beta\}}} E_{i,j}(y_i,y_j^t;x)}_{\text{unary}} \\
+ & \underbrace{\sum_{\substack{(i,j)\in E,\\ y_i^t\notin\{\alpha,\beta\},y_j^t\in\{\alpha,\beta\}}} E_{i,j}(y_i^t,y_j;x)}_{\text{unary}} + \underbrace{\sum_{\substack{(i,j)\in E,\\ y_i^t\in\{\alpha,\beta\},y_j^t\in\{\alpha,\beta\}}} E_{i,j}(y_i,y_j;x)}_{\text{pairwise}} \Bigg].
\end{aligned}
\tag{4.22}
$$

When dropping the constant terms and combining the unary terms, problem (4.22) is simplified and can be solved as a network flow problem [14] on a specially constructed auxiliary graph, with structure as shown in Figure 4.11.

The directed graph $G' = (V',\mathcal{E}')$ with non-negative edge weights t_i^α, t_i^β, and $n_{i,j}$ is constructed as follows.

$$
\begin{aligned}
V' &= \{\alpha,\beta\} \cup \{i \in V : y_i \in \{\alpha,\beta\}\},\\
E' &= \{(\alpha,i,t_i^\alpha) : \forall i \in V : y_i \in \{\alpha,\beta\}\}\\
&\quad \cup \{(i,\beta,t_i^\beta) : \forall i \in V : y_i \in \{\alpha,\beta\}\}\\
&\quad \cup \{(i,j,n_{i,j}) : \forall (i,j),(j,i) \in E : y_i,y_j \in \{\alpha,\beta\}\}.
\end{aligned}
$$

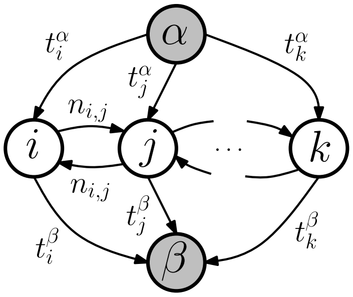


Fig. 4.11 Directed edge-weighted auxiliary graph construction. The linear min-cut in this graph corresponds to the optimal energy configuration in $\mathcal{N}_{\alpha,\beta}(y,\alpha,\beta)$.

The edge weights are calculated as follows.

$$n_{i,j} = E_{i,j}(\alpha,\beta;x), \tag{4.23}$$

$$t_i^{\alpha} = E_i(\alpha;x) + \sum_{\substack{(i,j)\in\mathcal{E},\\ y_j\notin\{\alpha,\beta\}}} E_{i,j}(\alpha,y_j;x), \tag{4.24}$$

$$t_i^{\beta} = E_i(\beta;x) + \sum_{\substack{(i,j)\in\mathcal{E},\\ y_j\notin\{\alpha,\beta\}}} E_{i,j}(\beta,y_j;x). \tag{4.25}$$

Finding a directed minimum α–β-cut, that is, a cut which separates α and β in the graph G', solves (4.22). To see how this is possible, consider the cut shown in Figure 4.12. The value $f(\mathcal{C})$ of a cut $\mathcal{C}$ is the sum of the directed edge weights it cuts. For the example graph this would be

$$\begin{aligned} f(\mathcal{C}) &= t_i^{\alpha} + n_{i,j} + t_j^{\beta} + t_k^{\beta} \\ &= E_i(\alpha;x) + \sum_{\substack{(i,s)\in E,\\ y_s^t\notin\{\alpha,\beta\}}} E_{i,s}(\alpha,y_s^t;x) + E_{i,j}(\alpha,\beta;x) \\ &\quad + E_j(\beta;x) + \sum_{\substack{(i,s)\in E,\\ y_s^t\notin\{\alpha,\beta\}}} E_{i,s}(\beta,y_s^t;x) \\ &\quad + E_k(\beta;x) + \sum_{\substack{(k,s)\in E,\\ y_s^t\notin\{\alpha,\beta\}}} E_{k,s}(\beta,y_s^t;x), \end{aligned}$$

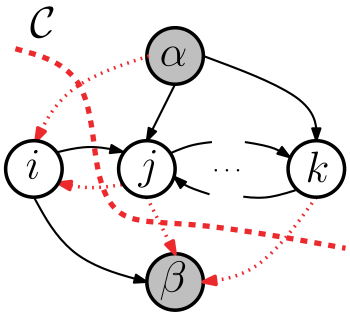


Fig. 4.12 A minimum α–β-cut $\mathcal{C}$ and its directed edge cutset (shown dotted, in red).

which corresponds exactly to (4.22) for $y_i = \alpha$, $y_j = \beta$, and $y_k = \beta$. This holds in general and [28] showed that the optimal labeling can be constructed from the α–β-mincut $\mathcal{C}$ as

$$y_i = \begin{cases} \alpha & \text{if } (\alpha, i) \in \mathcal{C}, \\ \beta & \text{if } (i, \beta) \in \mathcal{C}. \end{cases}$$

Because exactly one of the edges must be cut for $\mathcal{C}$ to be an α–β-cut, the min-cut exactly minimizes (4.22).

Solving the min-cut problem on the auxiliary graph G' can be done efficiently by using linear max-flow algorithms. For graphs such as the one shown in G' where all nodes are connected to the source- and sink-node, specialized max-flow algorithms with superior *empirical* performance have been developed, see Boykov and Kolmogorov [26]. The best known algorithms for linear max-flow problems have a computational complexity of $O(|V|^3)$ and $O(|V||\mathcal{E}|\log(|V|))$, see [14].

The α–β-swap neighborhood depends on two label parameters α and β and each combination of α and β induces a different neighborhood. Therefore, we typically cycle through all pairwise label combinations until no further improvement can be made, that is, the solution is optimal with respect to all α–β-swap neighborhoods.

Because the min-cut problem is solvable efficiently only if all edge weights are non-negative, it is now clear why the pairwise energies have to be semi-metric: this property guarantees non-negativity of all edge weights in the auxiliary graph G'.

4.5.2.2 α-expansion

While the α–β-swap neighborhood was defined on label pairs, the α-expansion neighborhood is defined on a single label. For a given label α, the α-expansion neighborhood $\mathcal{N}_\alpha(y, \alpha)$ allows every node to either remain in its current state *or* to change its state to α. Finding the optimal solution within the neighborhood of the current solution is again a binary labeling problem. However, in order to work it requires $E_{i,j}$ to satisfy the triangle inequality for all $(i, j) \in E$ and is thus more limited, compared to the α–β-swap.

Formally, the α-expansion neighborhood is defined as follows.

$$\mathcal{N}_\alpha : \mathcal{Y} \times \mathbb{N} \to 2^{\mathcal{Y}},$$
$$\mathcal{N}_\alpha(y,\alpha) := \{z \in \mathcal{Y} : \forall i \in V : z_i \in \{y_i,\alpha\}\}.$$

As for the α–β-swap neighborhood, Boykov et al. [28] showed that the minimizer within $\mathcal{N}_\alpha(y,\alpha)$ can be found by solving a network flow problem on a auxiliary graph whose edge weights can be derived by decomposing the energy function within the neighborhood.

$$\begin{aligned}
y^{t+1} &= \operatorname*{argmin}_{y \in \mathcal{N}_\alpha(y^t,\alpha)} E(y;x) \\
&= \operatorname*{argmin}_{y \in \mathcal{N}_\alpha(y^t,\alpha)} \sum_{i \in V} E_i(y_i;x) + \sum_{(i,j) \in \mathcal{E}} E_{i,j}(y_i,y_j;x) \\
&= \operatorname*{argmin}_{y \in \mathcal{N}_\alpha(y^t,\alpha)} \Bigg[\sum_{\substack{i \in V,\\ y_i = \alpha}} E_i(\alpha;x) + \sum_{\substack{i \in V,\\ y_i \neq \alpha}} E_i(y_i^t;x) \\
&\quad + \sum_{\substack{(i,j) \in \mathcal{E},\\ y_i = \alpha, y_j = \alpha}} E_{i,j}(\alpha,\alpha;x) + \sum_{\substack{(i,j) \in \mathcal{E},\\ y_i = \alpha, y_j \neq \alpha}} E_{i,j}(\alpha,y_j^t;x) \\
&\quad + \sum_{\substack{(i,j) \in \mathcal{E},\\ y_i \neq \alpha, y_j = \alpha}} E_{i,j}(y_i^t,\alpha;x) + \sum_{\substack{(i,j) \in \mathcal{E},\\ y_i \neq \alpha, y_j \neq \alpha}} E_{i,j}(y_i^t,y_j^t;x) \Bigg]. \qquad (4.26)
\end{aligned}$$

The graph structure of the auxiliary graph depends on the current solution y^t and is illustrated in Figure 4.13.

Formally, given $G = (V,\mathcal{E})$ and a current solution $y^t \in \mathcal{Y}$, the auxiliary directed, edge-weighted graph $G' = (V',\mathcal{E}')$ is constructed as follows.

$$\begin{aligned}
V' &= \{\alpha,\bar{\alpha}\} \cup V \cup \{\overline{ij} : \forall (i,j) \in \mathcal{E} : y_i^t \neq y_j^t\}, \\
E' &= \{(\alpha,i,t_i^\alpha) : \forall i \in V\} \cup \{(i,\bar{\alpha},t_i^{\bar{\alpha}}) : \forall i \in V\} \\
&\quad \cup \{(i,j,n_{i,j}),(j,i,n_{i,j}) : \forall (i,j) \in \mathcal{E} : y_i^t = y_j^t\} \\
&\quad \cup \{(\overline{ij},\bar{\alpha},t_{\overline{ij}}^{\bar{\alpha}}) : \forall (i,j) \in \mathcal{E} : y_i^t \neq y_j^t\} \\
&\quad \cup \{(i,\overline{ij},n_{i,\overline{ij}}),(\overline{ij},i,n_{i,\overline{ij}}),(j,\overline{ij},n_{j,\overline{ij}}),(\overline{ij},j,n_{j,\overline{ij}}) : \forall (i,j) \in \mathcal{E} : \\
&\qquad y_i^t \neq y_j^t\},
\end{aligned}$$

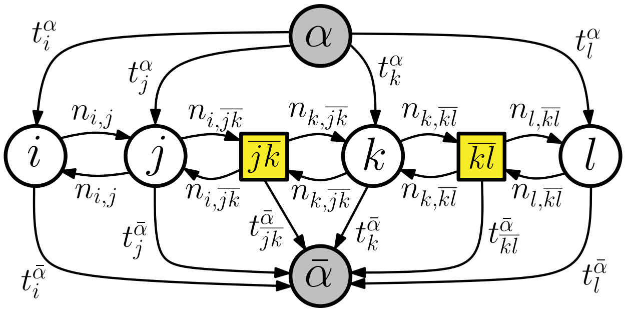


Fig. 4.13 Alpha expansion graph construction: all pixels i, j, k, and l are embedded into a graph and connected to a source node "α" and a sink node "$\bar{\alpha}$" (drawn in gray). For pairs of pixels $(i,j) \in \mathcal{E}$ which are currently labeled with different labels, $y_i^t \neq y_j^t$ a new node "$\overline{ij}$" is introduced (drawn squared). The minimum directed α–$\bar{\alpha}$ cut on this graph is the minimum energy solution in $\mathcal{N}_{\alpha}(y^t, \alpha)$.

with non-negative edge weights calculated from the current solution y^t as follows.

$$\begin{aligned}
t_i^{\alpha} &= E_i(\alpha; x), \\
t_i^{\bar{\alpha}} &= \begin{cases} \infty & \text{if } y_i^t = \alpha, \\ E_i(y_i^t; x) & \text{otherwise,} \end{cases} \\
n_{i,j} &= E_{i,j}(y_i^t, \alpha; x) \quad \left(= E_{i,j}(\alpha, y_j^t; x)\right), \\
t_{\overline{ij}}^{\bar{\alpha}} &= E_{i,j}(y_i^t, y_j^t; x), \\
n_{i,\overline{ij}} &= E_{i,j}(y_i^t, \alpha; x).
\end{aligned}$$

The min-cut on G' corresponds to the minimum in (4.26) by constructing y^{t+1} from the minimum weight edge cutset $\mathcal{C}$ of G' as

$$y_i^{t+1} = \begin{cases} \alpha & \text{if } (\alpha, i) \in \mathcal{C} \\ y_i^t & \text{otherwise} \end{cases},$$

for all $i \in V$. The analysis and proof can be found in [28]. The requirement that $E_{i,j}$ must satisfy the triangle inequality is needed to show that cuts like the one shown in Figure 4.14 cannot be minimal. If the triangle inequality holds, then the cut cannot be minimal as cutting

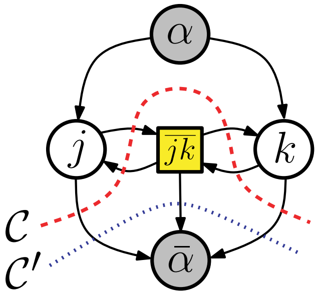


Fig. 4.14 A cut $\mathcal{C}$ of the shown type (drawn dashed, in red) can never be a minimal cut in G'. The cut $\mathcal{C}'$ (drawn dotted, in blue) always has an energy no greater than $\mathcal{C}$, due to the triangle inequality assumption on the pairwise energy terms $E_{i,j}$.

$(\overline{jk}, \bar{\alpha})$ directly gives a lower energy:

$$\begin{aligned} E(\mathcal{C}) &= n_{j,\overline{jk}} + n_{k,\overline{jk}} + t_j^{\bar{\alpha}} + t_k^{\bar{\alpha}} \\ &= E_{j,k}(y_j^t, \alpha; x) + E_{j,k}(y_k^t, \alpha; x) + t_j^{\bar{\alpha}} + t_k^{\bar{\alpha}} \\ &\geq E_{j,k}(y_j^t, y_k^t; x) + t_j^{\bar{\alpha}} + t_k^{\bar{\alpha}} \\ &= t_{\overline{jk}}^{\bar{\alpha}} + t_j^{\bar{\alpha}} + t_k^{\bar{\alpha}} \\ &= E(\mathcal{C}'; x). \end{aligned}$$

As already done for the α–β-swap, the parameter α in the α-expansion is iterated over to obtain a solution that is optimal with respect to all α-expansion neighborhoods. In practice the α-expansion is often preferred over the α–β-swap because it converges faster and Boykov established a worst case bound on the energy with respect to the true optimal energy.[4]

Example 4.6 (Stereo Disparity Estimation with Graph Cuts). Given two color images that are taken from slightly shifted positions, it is possible to estimate the left–right disparity for each pixel. Together with a camera model this allows us to estimate for each pixel the distance from the observer. Two example input images are shown in

[4] One advantage of the α–β-swap algorithm besides its generality is that it can be easily parallelized by processing disjoint pairs (α_1,β_1), (α_2,β_2), $\alpha_2,\beta_2 \notin \{\alpha_1,\beta_1\}$ at the same time.

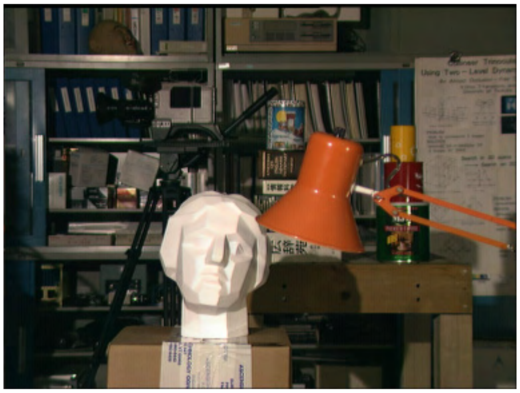


Fig. 4.15 Left input image.

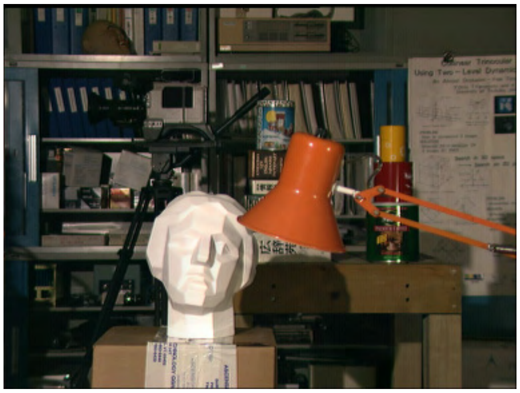


Fig. 4.16 Right input image.

Figures 4.15 and 4.16, and the ground truth per-pixel disparities to be predicted are shown in Figure 4.18.

Birchfield and Tomasi [18] proposed to perform a simple per-pixel disparity estimation by locally matching a block of pixels between the two images. Each variable y_i takes values from a set of disparities, typically chosen to be $\{0,1,\dots,K-1,K\}$. As this local estimate will in general be very noisy, Boykov et al. [27] use an MRF to encourage a piecewise smooth estimate of disparity values. The resulting energy has unary energy terms from the local pixel cost, as well as pairwise energies that encourage a smooth solution. The pairwise energies can be constructed as to fit the metric assumption made by graph cut methods.

Two studies comparing graph cuts with other MAP inference methods, Tappen and Freeman [147] and Szeliski et al. [145], found that for this problem and energies, the graph cut solutions are among the most accurate, and moreover the graph cut method is computationally efficient. An example of the output produced by α-expansion is shown in Figures 4.17.

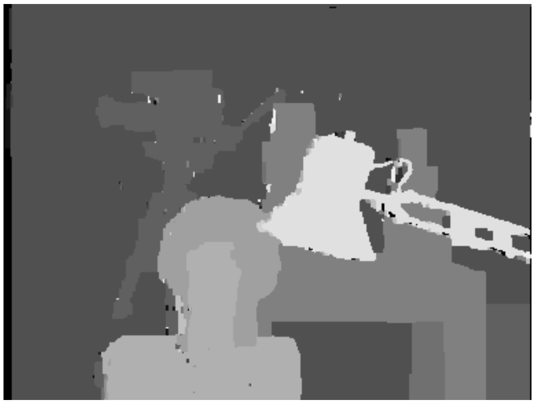


Fig. 4.17 Approximate MAP state from α-expansion.

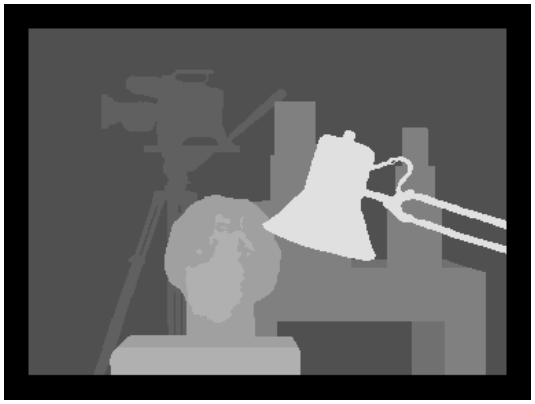


Fig. 4.18 Ground truth disparity map.

4.5.2.3 Limitations and extensions of graph cut inference

The efficiency of graph cut based energy minimization algorithms has lead to a flurry of research into this direction. We give a brief overview of the main results and research directions.

Ishikawa [62] gives a characterization of energies representable as graph cut problems for the case of multilabel states, i.e., where $|\mathcal{Y}_i| > 2$ for some $i \in V$. In general, to characterize solvable energies with high-order interactions is ongoing research. Kohli et al. [74, 75] gave an example of an energy term with simple structure, called the $\mathcal{P}^n$ generalized Potts potential which can be optimized using graph cuts. Ramalingam et al. [122] applied these $\mathcal{P}^n$ interactions to improve image segmentation results.

For energies which do not satisfy regularity conditions, Kolmogorov and Rother [79] give a graphcut-based iterative algorithm, QPBO, that uses *probing* techniques from combinatorial optimization, producing an approximate minimizer. In case the nodes have only binary states, the algorithm enjoys a favorable *partial optimality* property: all node states determined by the algorithm are either certain or uncertain with the guarantee that there exists an optimal solution which, when considering the certain nodes only, is identical to the solution provided by the algorithm.

Another research direction has been to improve the efficiency of graphcut based minimization algorithms. For planar graph structures common in computer vision progress has been made by using efficient network flow algorithms specific to planar graphs, see Schraudolph and Kamenetsky [137] and Schmidt et al. [133]. For general graphs with multilabel states, the most efficient current graph cut algorithms are due to Alahari et al. [3] and Komodakis et al. [83]. Both algorithms reuse computations from previous iterations.

4.5.3 Model Reduction

A natural and widely applicable technique to reduce the complexity of the prediction problem is to view the optimization problem as making many inter-dependent decisions. By *fixing* a subset of these decisions *a priori* the problem size is reduced and the remaining problem can be solved exactly.

A simple method to reduce the model is to fix two or more decisions to be equal, representing them by only a single variable. *Superpixels* introduced in [124, 107] are a common example of this reduction for image labeling tasks: by representing large sets of pixels as a single superpixel, a set of decisions — assigning one label to each pixel — is reduced to the single decision of assigning a label to the superpixel, as shown in Figures 4.19 and 4.20.

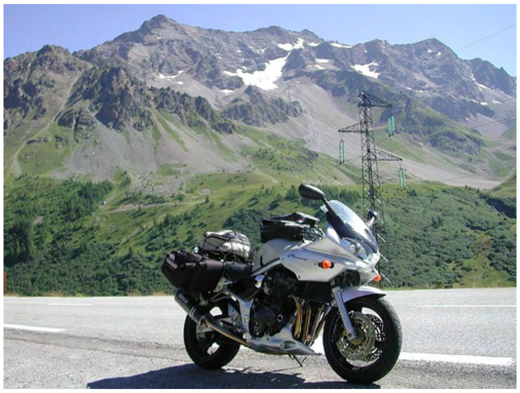


Fig. 4.19 Input image: 500 × 375 pixels, for a total of 187,500 labeling decisions.

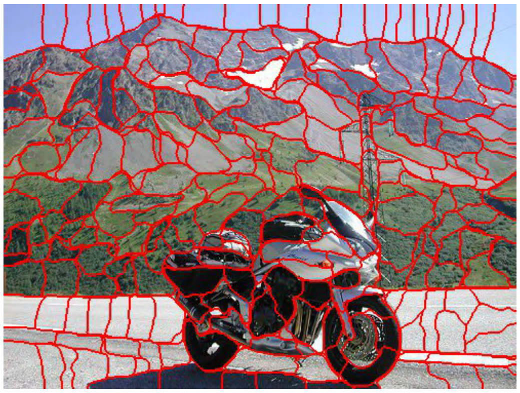


Fig. 4.20 The same image with 149 superpixels and hence 149 decisions.

By grouping individual decisions into joint decisions we effectively constrain the feasible set $\mathcal{Y}$ to a subset $\mathcal{Y}' \subset \mathcal{Y}$. Therefore the optimal solution $y^* \in \mathcal{Y}$ might no longer be identifiable and we have given up optimality.

The idea of grouping variables to obtain a smaller inference problem has been explored further by [70].

4.6 Giving up Worst-case Complexity

The worst-case complexity of solving a problem exactly is inherently pessimistic: from the set of all possible problem instances only the most difficult ones determine the worst-case complexity. Real problem instances might not at all show the special properties that make the problem difficult or at least show them only to a smaller extend. As an example, although exact MAP inference in many image segmentation models is NP-hard, most often the unary interactions are strong enough to leave only very few local contradictions that have to be resolved.

Therefore an exact algorithm that is efficient on practical instances and exhibits the worst-case complexity only on the truly difficult instances can be acceptable. A possible disadvantage is that a priori the runtime of such an algorithm is hard to predict.

4.6.1 Branch-and-Bound

Branch and bound is a general scheme for constructing algorithms to solve exactly hard optimization problems in case $\mathcal{Y}$ is finite.[5] A branch and bound algorithm performs an *implicit enumeration* of $\mathcal{Y}$ and is guaranteed to find the optimal solution $y^* \in \mathcal{Y}$. The worst case complexity of a branch and bound algorithm is typically the same as an exhaustive enumeration of $\mathcal{Y}$ and therefore usually exponential in the problem size.

Despite having a bad worst case complexity, well designed branch and bound algorithms can be very efficient on relevant problem instances. Moreover, the general scheme makes few assumptions, such

[5] Branch and bound is a more general idea and is also applied for global optimization of continuous functions. The case of finite $\mathcal{Y}$ simplifies the description of the algorithm.

that considerable flexibility remains for incorporating problem-specific knowledge into the algorithm.

Branch and bound is a *divide and conquer* algorithm and can be described as follows [13, 32, 84]. At all times, the algorithm maintains a partitioning of $\mathcal{Y}$ into *active* and *closed* nodes. The closed nodes cover parts of $\mathcal{Y}$ that we have proven not to contain a solution better than the currently best known solution. In contrast, for the parts of $\mathcal{Y}$ covered by the active nodes we are not yet sure and infact they may contain the optimal solution. The partitioning is of the form $\mathcal{Y} = (A_1 \cup \cdots \cup A_k) \cup (C_1 \cup \cdots \cup C_l)$, as shown in Figure 4.21. Let $\mathcal{A} = \{A_1, \ldots, A_k\}$ be the set of *active* nodes, and $\mathcal{C} = \{C_1, \ldots, C_l\}$ be the set of *closed* nodes, such that $\mathcal{A} \cup \mathcal{C}$ partitions $\mathcal{Y}$. Initially we have $\mathcal{A} = \{\mathcal{Y}\}$ and $\mathcal{C} = \emptyset$, but during the course of the algorithm subsets of $\mathcal{Y}$ will be moved from the set of active nodes to the set of closed nodes. Eventually all elements have been moved and at the termination of the algorithm we have $\mathcal{A} = \emptyset$, $\mathcal{C} = \{\mathcal{Y}\}$, the reverse of the initial assignment. During the course of the algorithm this is done in such a way that the optimal solution y^* is identified.

To achieve this, the algorithm applies as two operations a *branching* and a *bounding* step. The bounding step maintains for each element $A \in \mathcal{A}$ an upper and lower bound on the optimal solution *restricted to* A, as

$$\underline{g}(x, A) \leq \max_{y \in A} g(x, y) \leq \overline{g}(x, A). \tag{4.27}$$

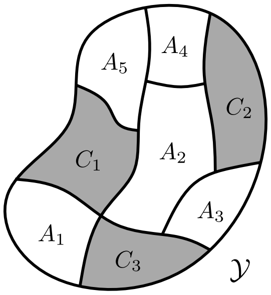


Fig. 4.21 Partitioning maintained by branch-and-bound: the overall set $\mathcal{Y}$ is represented by *active* subsets A_i (drawn in white) and *closed* subsets C_i (drawn shaded).

The branching step takes an element $A \in \mathcal{A}$ and removes it from $\mathcal{A}$. The element A is partitioned into two or more nonempty subsets, which are added to $\mathcal{A}$, as shown in Figure 4.22. Additionally for each subset a new lower and upper bound is computed. Typically, when evaluated on smaller sets the bounds become stronger and eventually if $A = \{y\}$ the bounds becomes exact, i.e., we have $\underline{g}(x, \{y\}) = g(x, y) = \overline{g}(x, \{y\})$. A strong enough bound for a subset $A \subseteq \mathcal{Y}$ allows *pruning* the entire set A: in case $\overline{g}(A)$ is smaller than or equal to the value of a known solution, then it can safely be moved from $\mathcal{A}$ to $\mathcal{C}$ as it cannot contain a better solution than the one we already know. This is shown in Figure 4.23. This above informal description is summarized in Algorithm 7 on page 270.

The algorithm is an abstract scheme and requires the definition of a problem-dependent selection, branching and bounding function.

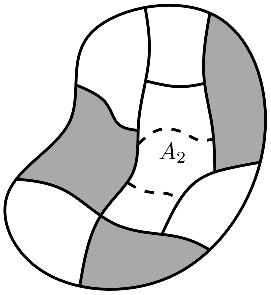


Fig. 4.22 Branching step: partitioning an active subset A into two or more subsets. Here the partitioning is into three subsets.

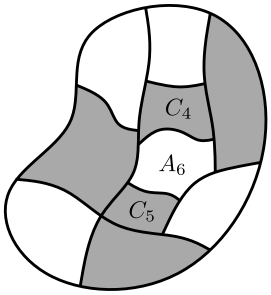


Fig. 4.23 Bounding step: some of the new subsets A_i can be closed by evaluating $\overline{g}(A_i)$.

Algorithm 7: Branch and Bound

```
$y^* = \textsc{BranchAndBound}(\mathcal{Y}, g, \overline{g}, \underline{g})$
**Input:**
    $x \in \mathcal{X}$ input data (observation)
    $\mathcal{Y}$ finite feasible set
    $g : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ objective function
    $\overline{g} : \mathcal{X} \times \mathcal{B} \to \mathbb{R}$ upper bound with $\overline{g}(x,B) \geq \max_{y \in B} g(x,y)$
    $\underline{g} : \mathcal{X} \times \mathcal{B} \to \mathbb{R}$ lower bound with $\underline{g}(x,B) \leq \max_{y \in B} g(x,y)$
**Output:**
    $y^* \in \mathcal{Y}$ global optimal solution with $y^* = \operatorname{argmax}_{y \in \mathcal{Y}}\, g(x,y)$
**Algorithm:**
$(L,U) \leftarrow (\underline{g}(x,\mathcal{Y}), \overline{g}(x,\mathcal{Y}))$ {Initialize global bounds on $g(x,y^*)$}
$(\mathcal{A},\mathcal{C}) \leftarrow (\{\mathcal{Y}\}, \emptyset)$
**while** $L < U$ **do**
   *Select* an element $A \in \mathcal{A}$
   $\mathcal{A} \leftarrow \mathcal{A} \setminus \{A\}$
   *Branch*: partition $A$ into $k \geq 2$ nonempty sets $A_1, \ldots, A_k$
   **for** $i = 1, \ldots, k$ **do**
      *Bound*: compute upper bound $\overline{g}(x,A_i)$
      *Bound*: compute lower bound $\underline{g}(x,A_i)$ and $y_i$ with $g(x,y_i) = \underline{g}(x,A_i)$
      **if** $\underline{g}(x,A_i) > L$ **then**
         $(y^*, L) \leftarrow (y_i, \underline{g}(x,A_i))$ {Increase global lower bound}
      **end if**
      $\mathcal{A} \leftarrow \mathcal{A} \cup \{A_i\}$
   **end for**
   $U \leftarrow \max_{A \in \mathcal{A}} \overline{g}(x,A)$ {Update global upper bound}
   $\mathcal{E} \leftarrow \{A \in \mathcal{A} : \overline{g}(x,A) \leq L$ or $\underline{g}(x,A_i) = \overline{g}(x,A_i)\}$ {Prune nodes}
   $(\mathcal{A},\mathcal{C}) \leftarrow (\mathcal{A} \setminus \mathcal{E}, \mathcal{C} \cup \mathcal{E})$
**end while**
```

In general, computing a bound efficiently might be feasible only for certain kinds of partitionings of $\mathcal{Y}$.

The selection step of the algorithm picks an $A \in \mathcal{A}$ to be partitioned. Ideally A is chosen such that it contains y^* but generally this information is not available and therefore the choice is based on available data such as the lower bound $\underline{g}(x,A)$ or the upper bound $\overline{g}(x,A)$. A common practise is to choose A as the element from $\mathcal{A}$ with the highest lower bound $\underline{g}(x,A)$ because an increase in L allows us to prune nodes from $\mathcal{A}$.

When designing the branching function it is important to partition a given subset of $\mathcal{Y}$ in such a way that the lower and upper bounds can be evaluated efficiently for the resulting disjoint subsets. Typically the lower and upper bounds can be evaluated only for some subsets of $\mathcal{Y}$ which we denote by $\mathcal{B} \subseteq 2^{\mathcal{Y}}$. The branching function will therefore only output sets $A_1,\dots,A_k$ that are elements of $\mathcal{B}$. In the examples below we will illustrate this point.

The bounding functions provide upper and lower bounds on the value of the optimal solution within a subset of $\mathcal{Y}$. The lower bound function $\underline{g}(x,A)$ additionally provides an element $y \in A$ realizing $\underline{g}(x,A) = g(x,y)$. Because any element in A is a lower bound on $\max_{y \in A} g(x,y)$, the construction of a trivial lower bounding function by picking a random element from A is valid. A stronger lower bound over A can be provided by choosing a better solution y according to $g(x,y)$. Therefore solution methods such as local search or other heuristics can be used for constructing the lower bound.

In contrast, the construction of a valid *upper* bound that can be evaluated efficiently is more difficult. In case the problem $\max_{y \in A} g(x,y)$ can be formulated as a mathematical programming problem, principled relaxation and decomposition techniques can be applied to obtain an upper bound, some of which we discuss in Section 4.7. An alternative approach to produce an upper bound is to analyze $g(x,y)$ over $y \in A$, developing an upper bound that can be computed in closed form. Below we will discuss one example for each approach.

Example 4.7(Efficient Subwindow Search). The *efficient subwindow search (ESS)* procedure proposed by Lampert et al. [89, 90] is a branch-and-bound method to locate within a given image $x \in \mathcal{X}$ a rectangle y^* that achieves a maximum classification score as measured

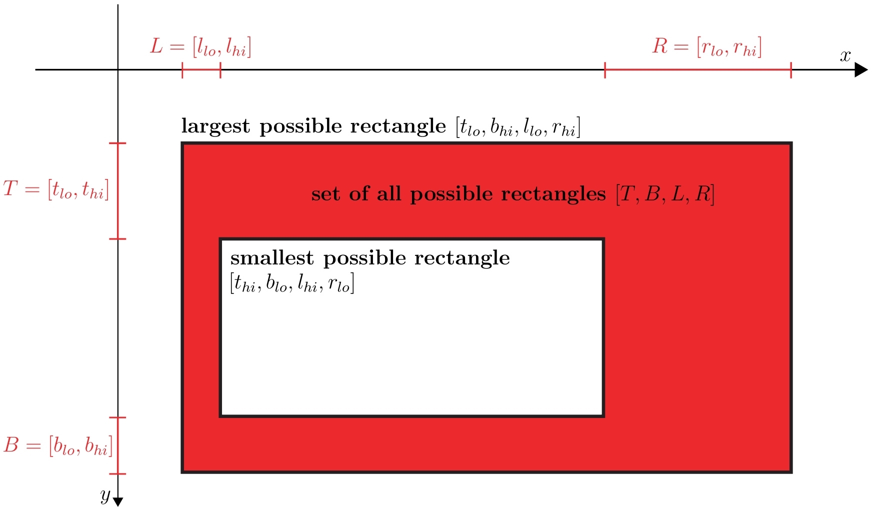

$L = [l_{lo}, l_{hi}]$
$R = [r_{lo}, r_{hi}]$
x
largest possible rectangle $[t_{lo}, b_{hi}, l_{lo}, r_{hi}]$
$T = [t_{lo}, t_{hi}]$
set of all possible rectangles $[T, B, L, R]$
smallest possible rectangle
$[t_{hi}, b_{lo}, l_{hi}, r_{lo}]$
$B = [b_{lo}, b_{hi}]$
y

Fig. 4.24 An ESS rectangle set specified by four intervals for the top, bottom, left, and right coordinates of a rectangle.

by an arbitrary classification function $g\colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$. Therefore, the set $\mathcal{Y}$ consists of all possible rectangles within the image. Lampert and Blaschko use $\mathcal{B} \subset 2^{\mathcal{Y}}$, that is, all sets of rectangles that can be selected by specifying four min–max intervals for each respective four coordinates: top, bottom, left, right. One element $B \in \mathcal{B}$ is visualized in Figure 4.24. The set $\mathcal{B}$ includes as elements all individual rectangles, i.e., $\{y\} \in \mathcal{B}$ for all $y \in \mathcal{Y}$.

For any element in $\mathcal{B}$ the branching step is realized by selecting one interval and splitting it into two halves.

Lampert et al. [89] show that for certain classification functions $g(x,y)$ an efficient upper bound $\overline{g}(x,y)$ can be constructed. One such case is where the classification function is the sum of weighted scores of the local image features falling into the rectangle, as is the case for the popular bag-of-visual-words representation. For the binary case we have

$$g(x,y) = \beta + \sum_{x_i \text{ within } y} w(x_i), \tag{4.28}$$

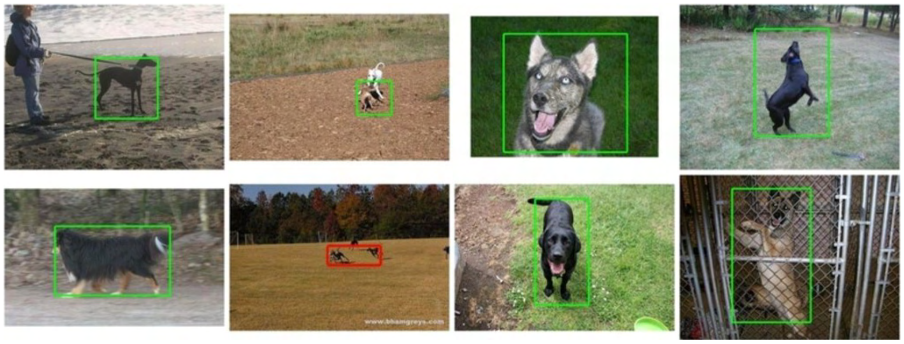


Fig. 4.25 Bounding boxes found using the efficient subwindow search branch-and-bound procedure on the `dog` class of the PASCAL VOC 2007 benchmark data set.

where $x_i \in \{1,2,\ldots,H\}$ is a set of quantized local image features and x_i within y is true if x_i falls into the image region specified by y. For any $B \in \mathcal{B}$ an upper bound on this score can be constructed as

$$\overline{g}(x,B) = \beta + \sum_{\substack{x_i \\ \text{within} \\ B_{\max}}} \max\{w(x_i),0\} + \sum_{\substack{x_i \\ \text{within} \\ B_{\min}}} \min\{0,w(x_i)\} \quad (4.29)$$

$$\geq \max_{y \in B} g(x,y), \quad (4.30)$$

where $B_{\max}$, $B_{\min}$ is the largest and smallest possible rectangle in B, respectively. The use of $\max\{w(x_i),0\}$ and $\min\{0,w(x_i)\}$ effectively sums only over the positive and negative terms, respectively. If $B = \{y\}$ we have $B_{\max} = B_{\min} = B$ and therefore the bound becomes exact: $\overline{g}(x,B) = g(x,y)$.

Branch-and-bound with the above bound on a learned classification function allows us to find the optimal rectangle $y^* \in \mathcal{Y}$ efficiently, despite having a worst case complexity of $O(u^4)$ where u is the number of possible coordinate values along one rectangle dimension. Some example results are shown in Figure 4.25.

Example 4.8 (Branch-and-mincut). Lempitsky et al. [93] consider binary image segmentation tasks where an energy function depends on the value of a nonlocal parameter $y \in \mathcal{Y}$, where $\mathcal{Y}$ is a large finite set. For any fixed $y \in \mathcal{Y}$, an energy to be minimized over binary segmentation masks $x \in \{0,1\}^V$ is used, which has the following form:

$$E(x,y) = C(y) + \sum_{p \in V} F^p(y) x_p + \sum_{p \in V} B^p(y)(1 - x_p) + \sum_{\{i,j\} \in \mathcal{E}} P^{pq}(y)|x_p - x_q|, \tag{4.31}$$

where $C(y)$ is a constant, F^p and B^p are unary energies for labeling pixel p as foreground $(x_p = 1)$ or background $(x_p = 0)$, respectively. A pairwise energy P^{pq} for pairs of pixels $\{p,q\} \in \mathcal{E}$ gives the cost for assigning p and q to different classes $(x_p \neq x_q)$. For a fixed y the energy (4.31) is submodular in x and can be efficiently minimized.

All energy terms depend on y, allowing nonlocal interactions such as shape priors. Lempitsky et al. demonstrate this by using $\mathcal{Y} = \Delta \times \Theta$, where Δ is a small set of prototype shapes and Θ is a discretized set of translation parameters. By defining

$$g(x,y) = \max_{x \in \{0,1\}^V} -E(x,y), \tag{4.32}$$

finding $y^* \in \mathcal{Y}$ effectively identifies *simultaneously* the best matching global shape and image segmentation mask, as shown in Figures 4.26 to 4.28.

For finding y^* Lempitsky et al. use the branch-and-bound framework, deriving an upper bound $\overline{g}(x,A) \geq \max_{y \in A} g(x,A)$ for any A by showing that

$$\max_{y \in A} g(x,y) \tag{4.33}$$

$$= \max_{y \in A} \max_{x \in 2^V} -E(x,y) \tag{4.34}$$

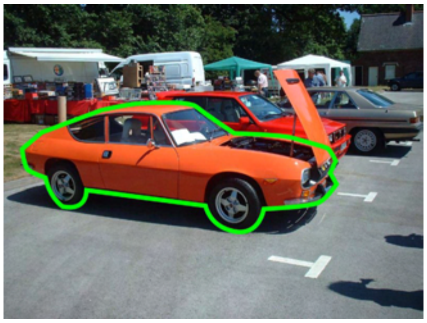


Fig. 4.26 An input image with shape prior mask overlay.

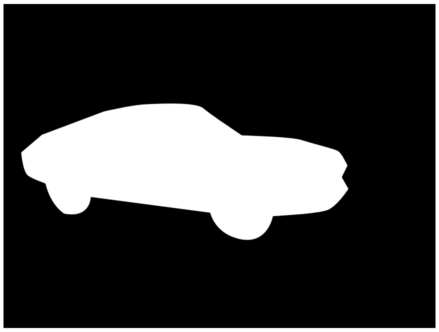


Fig. 4.27 Optimal element $y^* \in \mathcal{Y}$.

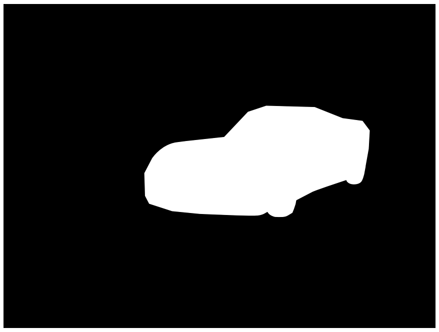


Fig. 4.28 Another element $y \in \mathcal{Y}$.

$$
\begin{aligned}
&= \max_{y \in A} \max_{x \in 2^V} - \Bigg[C(y) + \sum_{p \in V} F^p(y) x_p \\
&\quad + \sum_{p \in V} B^p(y)(1 - x_p) + \sum_{\{i,j\} \in \mathcal{E}} P^{pq}(y) |x_p - x_q| \Bigg] && (4.35) \\
&\leq \max_{x \in 2^V} \Bigg[\Big(\max_{y \in A} -C(y) \Big) + \sum_{p \in V} \Big(\max_{y \in A} -F^p(y) \Big) x_p \\
&\quad + \sum_{p \in V} \Big(\max_{y \in A} -B^p(y) \Big) (1 - x_p) && (4.36) \\
&\quad + \sum_{\{i,j\} \in \mathcal{E}} \Big(\max_{y \in A} -P^{pq}(y) \Big) |x_p - x_q| \Bigg] =: \overline{g}(x, A). && (4.37)
\end{aligned}
$$

The active node is chosen to be the node with the largest upper bound in each iteration. Branching is fixed by means of a pre-determined clustering in the parameter space.

Due to its popularity and flexibility, many variations and improvements to the branch-and-bound procedure exist. A popular modification is to use a fast heuristics algorithm to obtain a feasible solution for each node. Because any feasible solution bounds the objective of the optimal solution this can lead to substantial pruning of the search tree. If a suboptimal solution is sufficient, the branch-and-bound efficiency can be further improved by replacing the pruning step with

$$\mathcal{E} \leftarrow \{A \in \mathcal{A} : \overline{g}(x,A) \leq (L + \delta) \quad \text{or} \quad \underline{g}(x,A_i) = \overline{g}(x,A_i)\}, \tag{4.38}$$

where $\delta \geq 0$ is the accepted loss in optimality, i.e., the optimal solution y' returned by the new algorithm may no longer be optimal but is guaranteed to satisfy $g(x,y') \geq g(x,y^*) - \delta$. A more extensive discussion of branch-and-bound methods can be found in [32, 168].

The QPBO-P algorithm of Rother et al. [127] is similar in spirit to giving up worst-case complexity: although its worst-case runtime is still bounded by a polynomial, it depends on the problem instance. The QPBO-P method is an iterative energy minimization algorithm for binary pairwise random fields. For submodular pairwise energies the method requires only a single iteration, whereas in the nonsubmodular case only a partial solution is recovered. The labeled part of this incomplete partial solution is known to be optimal and therefore the problem is reduced to an optimization problem over the remaining unlabeled variables. By fixing each of these variables individually and observing the effect on the remaining unlabeled variables — a technique called *probing* — the set of labeled variables is grown until no further progress can be made and either the full optimal solution is recovered or a partial optimal labeling is returned.

4.7 Giving Up Integrality: Relaxations and Decompositions

The techniques described in this section are based on the fact that some hard optimization problems of the form (4.2) become *easier* when the feasible set is enlarged or the objective function is replaced by a bound.

The resulting modified problem is called *relaxation* because it is guaranteed to have a solution of no lower objective value — as measured by the original objective function — than the original problem. Formally, we define a relaxation as follows. The definition is due to Geoffrion [50].

Definition 4.2 (Problem Relaxation). Given two optimization problems $(g,\mathcal{Y},\mathcal{G})$ and $(h,\mathcal{Z},\mathcal{G})$ as by Definition 4.1, the problem $(h,\mathcal{Z},\mathcal{G})$ is said to be a *relaxation* of $(g,\mathcal{Y},\mathcal{G})$ if,

(1) $\mathcal{Z} \supseteq \mathcal{Y}$, i.e., the feasible set of the relaxation contains the feasible set of the original problem, and
(2) $\forall y \in \mathcal{Y}: h(x,y) \geq g(x,y)$, i.e., over the original feasible set the objective function h achieves no lower values than the objective function g.

An immediate consequence from the above definition is that by solving the relaxed problem $(h,\mathcal{Z},\mathcal{G})$, thus obtaining $z^* = \operatorname{argmax}_{z \in \mathcal{Z}} h(x,z)$, we also obtain an *upper bound* $h(x,z^*)$ on the objective $g(x,y^*)$ of the true problem, that is we have $h(x,z^*) \geq g(x,y^*)$.

The construction and solution of a relaxation is in contrast to the previous solution strategy in which we gave up optimality, leading us to find a minimizer over a *subset* $\mathcal{Y}' \subseteq \mathcal{Y}$ of the true feasible set $\mathcal{Y}$. Now we instead find the global maximizer over an enlarged set $\mathcal{Z} \supseteq \mathcal{Y}$.

The relaxation approach has some unique advantages and drawbacks compared to other approaches. One advantage is that the relaxed problem is usually in a problem class for which polynomial-time complexity bounds can be given and hence the relaxed problem can be solved optimally and efficiently. The latter property typically ensures that the optimal solutions of the relaxation are stable with respect to perturbations in learned model parameters [114]. This stability is particularly important when the relaxation is used for solving prediction problems during parameter learning [39, 86, 99] and other approximate solution approaches do not come with this guarantee.

One disadvantage of solving a relaxation instead of the original problem is that the obtained solution z^* might be outside the original feasible set, i.e., we have $z^* \in \mathcal{Z} \setminus \mathcal{Y}$. Depending on the particular problem, the meaning of z^* might then be unclear and heuristics have to be used to obtain a solution $y \in \mathcal{Y}$ that is "similar" to z^*.

We now discuss three principled techniques how relaxations can be constructed: integer programming, Lagrangian relaxation and Lagrangian decomposition.

4.7.1 Linear Programming Relaxations

A general method for constructing a relaxation is to start from an exact formulation to the problem in which the feasible set $\mathcal{Y}$ is specified by a set of constraints. By removing one or more constraints which make solving the problem difficult we can enlarge the feasible set and obtain a simplified problem.

For the case of linear programming relaxations the problem is first formulated as *integer linear program* (ILP) of the following form:

$$\max_y \; c^\top y \tag{4.39}$$

$$\text{sb.t. } Ay \leq b, \tag{4.40}$$

$$y \text{ is integer.} \tag{4.41}$$

In the above formulation we have a continuous decision domain $\mathcal{G} = \mathbb{R}^d$ and — if the polyhedron specified by $Ay \leq b$ is bounded — a finite feasible set $\mathcal{Y} = \{y \in \mathcal{G} \colon Ay \leq b, y \text{ is integer}\}$.

Because integer linear programs [168] are a very general model class, formulating a problem as integer program is often possible even for nonlinear objective functions and complicated finite feasible sets by introducing additional "auxiliary" variables [166].

By removing the integer constraint (4.41) from problem (4.39) we obtain the following *linear programming relaxation*.

$$\max_y \; c^\top y \tag{4.42}$$

$$\text{sb.t. } Ay \leq b. \tag{4.43}$$

The effect of the relaxation is illustrated in Figures 4.29 and 4.30.

In general the feasible set of the relaxation is a polyhedron and therefore a continuous set. When optimizing a linear function $c^\top y$ over

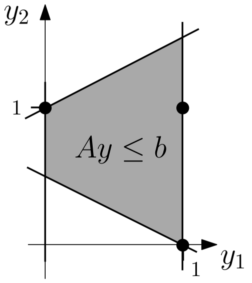


Fig. 4.29 The integer program solution set is given as the intersection of a polyhedron $\{y \in \mathbb{R}^2 \colon Ay \leq b\}$ and the integer lattice, resulting in a finite feasible set $\{(1,0),(0,1),(1,1)\}$.

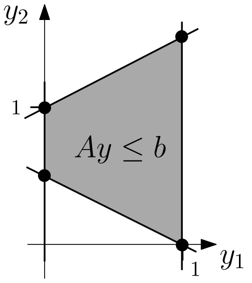


Fig. 4.30 The linear programming relaxation retains the polyhedral set $\{y \in \mathbb{R}^2 : Ay \leq b\}$ but does not require the solutions to be integral. The vertices of the polyhedron are shown.

a bounded polyhedron, it is enough to consider the *vertices* because at least one optimal solution to (4.42) is guaranteed to be a vertex. The vertices of (4.43) are shown in Figure 4.30. Some or all of the solutions of (4.39) might remain as vertices of the relaxed feasible set, some might be lost and there might be new vertices that do not lie on the integer lattice. These solutions are said to be *fractional*. The number of integral solutions cannot increase when relaxing the problem.

Example 4.9 (MAP–MRF Linear Programming Relaxation). In earlier sections discrete Markov random fields have been discussed as popular model for interdependent variables. We now discuss a linear programming relaxation for obtaining an approximate MAP configuration of a given random field model. Linear programming problems can be solved in polynomial time [15].

The following linear programming relaxation has first been proposed by Schlesinger in the 1970s [131] and has been extensively analyzed by Wainwright and Jordan [160] and Werner [162]. In the example below we only consider unary and pairwise factors, but there also exist straightforward extensions of the linear program to higher order factors, such as the one given in [163].

The Markov random field is defined by means of a *factor graph* and we define for each factor node one vector of binary variables. In Figure 4.31 we have $\mu_1 \in \{0,1\}^{\mathcal{Y}_1}$, $\mu_2 \in \{0,1\}^{\mathcal{Y}_2}$, and $\mu_{1,2} \in \{0,1\}^{\mathcal{Y}_1 \times \mathcal{Y}_2}$. For each labeling of the variables $Y_1 = y_1$ and $Y_2 = y_2$, we can set $\mu_1(y_1) = 1$, $\mu_2(y_2) = 1$, and $\mu_{1,2}(y_1,y_2) = 1$, and all other variables to zero. The energy values for each factor are given by means of tables $\theta_1 \in \mathbb{R}^{\mathcal{Y}_1}$, $\theta_2 \in \mathbb{R}^{\mathcal{Y}_2}$, and $\theta_{1,2} \in \mathbb{R}^{\mathbb{Y}_1 \times \mathbb{Y}_2}$. For any configuration $y \in \mathcal{Y}$, the *energy* $E(y)$ can then be computed as

$$\begin{aligned} E(y) &= \langle \mu, \theta \rangle & (4.44) \\ &= \sum_{y_1 \in \mathcal{Y}_1} \theta_1(y_1)\mu_1(y_1) + \sum_{y_2 \in \mathcal{Y}_2} \theta_2(y_2)\mu_2(y_2) \\ &\quad + \sum_{(y_1,y_2) \in \mathcal{Y}_1 \times \mathcal{Y}_2} \theta_{1,2}(y_1,y_2)\mu_{1,2}(y_1,y_2), & (4.45) \end{aligned}$$

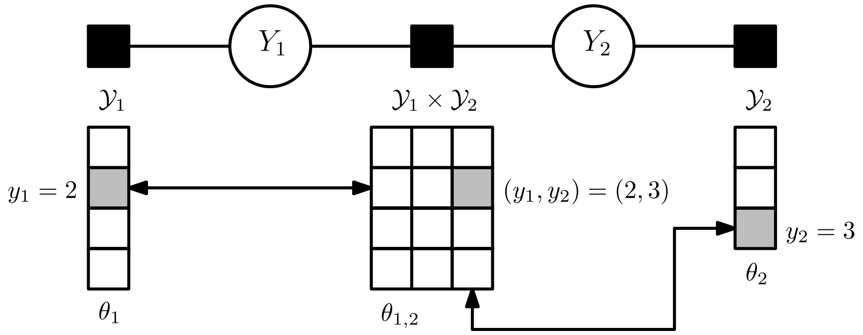


Fig. 4.31 A discrete Markov random field defined through a factor graph. Each factor node (drawn as ■) is defined over the product set of its adjacent variables node (drawn as ◯). In the figure, $|\mathcal{Y}_1| = 4$ and $|\mathcal{Y}_2| = 3$, hence the pairwise factor connecting to both variables Y_1 and Y_2 is a table of energy values indexed by elements from $\mathcal{Y}_1 \times \mathcal{Y}_2$. In the shown example, the overall energy of a configuration $(Y_1 = 2, Y_2 = 3)$ is the sum over three selected energy values (gray-shaded cells), one from each factor: $E(y_1,y_2) = \theta_1(y_1) + \theta_{1,2}(y_1,y_2) + \theta_2(y_2)$.

where we denote by μ and θ the concatenation of all variables and energy values, respectively. Not all possible assignments of μ_1, μ_2, and $\mu_{1,2}$ are possible, in fact, $\mu_{1,2}$ is completely determined by the choice of μ_1 and μ_2. Vice verse, given $\mu_{1,2}$, we can determine μ_1 and μ_2 as the column- and row-sum, respectively. This yields the two conditions

$$\mu_1(y_1) = \sum_{y_2 \in \mathcal{Y}_2} \mu_{1,2}(y_1,y_2), \quad \forall y_1 \in \mathcal{Y}_1, \tag{4.46}$$

$$\mu_2(y_2) = \sum_{y_1 \in \mathcal{Y}_1} \mu_{1,2}(y_1,y_2), \quad \forall y_2 \in \mathcal{Y}_2. \tag{4.47}$$

By finding among all feasible configurations of variable settings the one that minimizes the energy $\langle \mu, \theta \rangle$ we can solve the MAP–MRF problem. This is exactly what the following integer linear programming problem does for general factor graphs with unary and pairwise factors.

$$\min_{\boldsymbol{\mu}} \sum_{i \in V} \sum_{y_i \in \mathcal{Y}_i} \theta_i(y_i)\mu_i(y_i) + \sum_{\substack{\{i,j\} \\ \in E}} \sum_{\substack{(y_i,y_j) \in \\ \mathcal{Y}_i \times \mathcal{Y}_j}} \theta_{i,j}(y_i,y_j)\mu_{i,j}(y_i,y_j) \tag{4.48}$$

$$\text{sb.t.} \sum_{y_i \in \mathcal{Y}_i} \mu_i(y_i) = 1, \quad \forall i \in V, \tag{4.49}$$

$$\sum_{y_j \in \mathcal{Y}_j} \mu_{i,j}(y_i,y_j) = \mu_i(y_i), \quad \forall \{i,j\} \in E, \forall y_i \in \mathcal{Y}_i \tag{4.50}$$

$$\mu_i(y_i) \in \{0,1\}, \quad \forall i \in V, \forall y_i \in \mathcal{Y}_i, \tag{4.51}$$

$$\mu_{i,j}(y_i,y_j) \in \{0,1\}, \quad \forall \{i,j\} \in E, \forall (y_i,y_j) \in \mathcal{Y}_i \times \mathcal{Y}_j. \tag{4.52}$$

It is known that if the factor graph is tree-structured or if all pairwise interactions are submodular, then the linear programming relaxation obtained by relaxing (4.51) and (4.52) is tight, that is, it has an integral optimal solution as shown in [160]. One such case is given in Example 4.2, where the pairwise terms are all submodular.

In some cases, multiple integer linear programming formulations can be combined to obtain tractable relaxations, as the following example shows.

Example 4.10 (Random Fields with Connectivity Constraint). In [115] the problem of binary image segmentation with side constraints is addressed. The constraint of interest is to ensure that the segmentation mask is topologically connected. This is a meaningful constraint if a single unoccluded object is to be segmented. Unfortunately it is not possible to decompose the constraint into low-order factors. Instead, the authors propose an integer linear programming formulation with exponentially many linear inequality constraints of the form:

$$\mu_i(1) + \mu_j(1) - \sum_{k \in S} \mu_k(1) \leq 1, \tag{4.53}$$

where i and j are two image locations and S is any set of vertices whose removal disconnect i and j. Therefore, if i and j are labeled with the foreground label, then at least one pixel $k \in S$ must also be labeled foreground. This is illustrated in Figure 4.32. Although the number of constraints is exponential in the number of pixels, it is still possible to optimize a relaxation.

By combining this tractable relaxation with the above MAP–MRF LP relaxation the authors approximate the constrained image segmentation problem.

Related approximations to the connectivity constrained segmentation problem have been proposed by Vicente et al. [157] and Chen et al. [31] using graph cuts, and Lempitsky et al. [94] using linear programming relaxations.

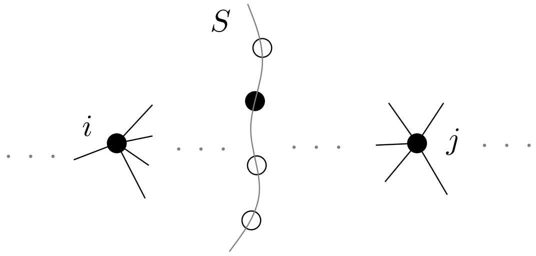


Fig. 4.32 Effect of the inequality (4.53): if pixel i is labeled foreground and pixel j is labeled foreground, then at least one pixel in every separating set S must also be labeled foreground.

4.7.2 Lagrangian Relaxation

Instead of dropping a constraint completely in order to construct a relaxation — as done when relaxing an integer linear program to a linear program, an alternative method is Lagrangian relaxation [53, 92]. In Lagrangian relaxation, one or more of the constraints are incorporated into a new *parameterized* objective function. For each parameter setting, solving for the maximum of the new objective function under the remaining constraints provides an *upper bound* on the optimal solution value of the original problem. By changing the parameter, another, possibly better bound can be obtained. Finding the smallest upper bound finds the strongest possible relaxation.

Formally, Lagrangian relaxation is applicable in case the feasible set is the intersection of two sets, i.e., $\mathcal{Y} = \mathcal{D} \cap \mathcal{C}$, resulting in the following problem.

$$\max_{y} \; g(x,y) \tag{4.54}$$

$$\text{sb.t. } y \in \mathcal{D}, \tag{4.55}$$

$$y \in \mathcal{C}. \tag{4.56}$$

The assumption made is that optimizing $g(x,y)$ over $\mathcal{D}$ is "easy," either because the solution could be constructed trivially or because an efficient method is available, but that optimizing over $\mathcal{D} \cap \mathcal{C}$ is difficult. To apply Lagrangian relaxation to problem (4.54) the set $\mathcal{C}$ needs to be expressed in terms of equality and inequality constraints as

$$\mathcal{C} = \{y \in \mathcal{G} : u_i(y) = 0, \forall i = 1,\ldots,I,\; v_j(y) \leq 0, \forall j = 1,\ldots,J\}, \tag{4.57}$$

where $u_i : \mathcal{G} \to \mathbb{R}$ are the equality constraint functions and $v_j : \mathcal{G} \to \mathbb{R}$ are the inequality constraint functions. All functions u_i, v_j must be differentiable. One typical additional assumption made is that each u_i is an affine function on $\mathcal{G}$ and each v_j is a convex function on $\mathcal{G}$. Together these assumptions guarantee that $\mathcal{C}$ is a convex subset of $\mathcal{G}$. With the expression of $\mathcal{C}$, problem (4.54) can be equivalently reformulated as the problem

$$\max_{y} \; g(x,y) \tag{4.58}$$

$$\text{sb.t. } y \in \mathcal{D}, \tag{4.59}$$

$$u_i(y) = 0, \quad i = 1,\dots,I, \tag{4.60}$$

$$v_j(y) \le 0, \quad j = 1,\dots,J. \tag{4.61}$$

We introduce *Lagrange multipliers* $\lambda_i \in \mathbb{R}$, $\mu_j \in \mathbb{R}_+$ associated to each constraint (4.60) and (4.61), respectively. This allows us to form the following *partial* Lagrangian problem in which the constraint $y \in \mathcal{D}$ is retained, but the constraints associated to $y \in \mathcal{C}$ are *dualized.*

$$q(\lambda,\mu) := \max_{y} \; g(x,y) + \lambda^\top u(y) + \mu^\top v(y) \tag{4.62}$$

$$\text{sb.t. } y \in \mathcal{D}. \tag{4.63}$$

The following theorem justifies the relaxation and the importance of problem (4.62).

Theorem 4.2 (Weak Duality of Lagrangian Relaxation). For differentiable functions $u_i\colon\mathcal{G} \to \mathbb{R}$ and $v_j\colon\mathcal{G} \to \mathbb{R}$, and for any $\lambda \in \mathbb{R}^I$ and any non-negative $\mu \in \mathbb{R}^J$, $\mu \ge 0$, we have that the optimal value of problem (4.62) is greater than or equal to the optimal value of (4.54). For a proof, see [13, Section 5.5.3].

For each pair of Lagrangian multipliers (λ,μ), solving problem (4.62) gives a different bound. Finding the lowest possible bound is itself an optimization problem, the so called *dual problem.* Let $q(\lambda,\mu)$ denote the value of (4.62) for a given pair (λ,μ). Then, solving

$$\min_{\lambda,\mu} \; q(\lambda,\mu) \tag{4.64}$$

$$\text{sb.t. } \mu \ge 0 \tag{4.65}$$

yields the strongest upper bound. In case $q(\lambda,\mu)$ can be evaluated efficiently, problem (4.64) can be solved optimally using the following result.

Theorem 4.3 (Lagrangian Dual Function). The function q is *convex* in λ and μ, such that problem (4.64) is a convex, not necessarily differentiable minimization problem. In case q is unbounded below, then the original problem (4.54) is infeasible.

For any given λ, $\mu \geq 0$, let $y(\lambda,\mu) = \mathrm{argmax}_{y \in \mathcal{D}}(g(x,y) + \lambda^\top u(y) + \mu^\top v(u))$ denote the maximizer obtained when evaluating $q(\lambda,\mu)$. Then, a *subgradient* of q can be constructed by evaluating the constraint functions at $y(\lambda,\mu)$ as

$$u(y(\lambda,\mu)) \in \frac{\partial}{\partial \lambda} q(\lambda,\mu), \quad \text{and} \quad v(y(\lambda,\mu)) \in \frac{\partial}{\partial \mu} q(\lambda,\mu), \tag{4.66}$$

where $\frac{\partial}{\partial \lambda} q(\lambda,\mu)$ and $\frac{\partial}{\partial \mu} q(\lambda,\mu)$ denote the *subdifferentials* of $q(\lambda,\mu)$.

The subgradient result in Theorem 4.3 is a special case of *Danskin's theorem* [13, Proposition B.25].

Theorem 4.3 allows us to use standard optimization methods for nondifferentiable convex minimization [23] to solve for an optimal (λ^*,μ^*) that minimizes the upper bound. One simple and popular method is a generalization of gradient descent known as *subgradient method*, shown in Algorithm 8 and originally proposed by Shor [139]. The notation $[\cdot]_+$ projects a vector on the non-negative orthant by setting all its negative elements to zero.

Algorithm 8: Subgradient Method

```
$(\lambda^*,\mu^*) = \textsc{SubgradientMethod}(\lambda^0,\mu^0,T)$
Input:
    $\lambda^0 \in \mathbb{R}^I$ initial Lagrange multiplier related to $u$
    $\mu^0 \in \mathbb{R}^J$ initial Lagrange multiplier related to $v$
    $T \in \mathbb{N}$ number of iterations
Output:
    $(\lambda^*,\mu^*) \in \mathbb{R}^I \times \mathbb{R}^J_+$ approximate solution to (4.64)
Algorithm:
for $t = 0,1,\ldots,(T-1)$ do
    Obtain $q(\lambda^t,\mu^t)$ and $y(\lambda^t,\mu^t)$ by solving (4.62)
    Choose step size $\alpha^t > 0$
    $\lambda^{t+1} \leftarrow \lambda^t - \alpha^t u(y(\lambda^t,\mu^t))$
    $\mu^{t+1} \leftarrow [\mu^t - \alpha^t v(y(\lambda^t,\mu^t))]_+$
end for
$(\lambda^*,\mu^*) \leftarrow (\lambda^T,\mu^T)$
```

There are multiple options in how the step size α^t can be chosen in each iteration, and we only mention two popular choices. Further discussion can be found in [13, Section 6.3.1]. For step sizes satisfying the *diminishing step size* conditions, $\alpha^t \to 0$ and $\sum_{t=0}^{\infty} \alpha^t \to \infty$, convergence is guaranteed. The common step size choices are

(1) Simple diminishing step size,

$$\alpha^t = \frac{1+m}{t+m}, \tag{4.67}$$

where $m > 0$ is an arbitrary constant. This step size is simple to implement and reasonably effective when m is tuned to the problem.

(2) Polyak's step size,

$$\alpha^t = \beta^t \frac{q(\lambda^t,\mu^t) - \hat{q}}{\|u(y(\lambda^t,\mu^t))\|^2 + \|v(y(\lambda^t,\mu^t))\|^2}, \tag{4.68}$$

where $0 < \beta^t < 2$ is a diminishing step size (such as the first choice), and $\hat{q} \leq q(\lambda^*,\mu^*)$ is a lower bound on the optimal dual objective. Often a simple valid bound can be established by constructing a suboptimal $y \in \mathcal{Y}$ and taking $\hat{q} = g(x,y)$ because by Theorem 4.2 we have $g(x,y) \leq q(\lambda^*,\mu^*)$. In case no bound is (yet) available, the numerator of (4.68) can be set to a constant, as is done in the simple diminishing step size.

Before we give a practical example how Lagrangian relaxation can be fruitfully applied, let us discuss one more issue: primal solution recovery. Solving for (λ^*,μ^*) of (4.64) provides us with an upper bound $q(\lambda^*,\mu^*) \geq g(x,y^*)$ on the *value* of the solution y^* to (4.54), but unfortunately does not provide us with y^* itself. During the course of the subgradient method many candidates $y(\lambda^t,\mu^t)$ are produced, but they might violate the requirement $y \in \mathcal{C}$ or might be suboptimal. The following theorem provides a sufficient condition to identify whether a candidate is indeed an optimal solution.

Theorem 4.4 (Sufficient Optimality Conditions). If for a given λ, $\mu \geq 0$, we have $u(y(\lambda,\mu)) = 0$ and $v(y(\lambda,\mu)) \leq 0$ (primal feasibility) and further we have

$$\lambda^\top u(y(\lambda,\mu)) = 0, \quad \text{and} \quad \mu^\top v(y(\lambda,\mu)) = 0, \tag{4.69}$$

(complementary slackness), then $y(\lambda,\mu)$ is an optimal solution to (4.54) and (λ,μ) is an optimal solution to (4.64). See [23].

Theorem 4.4 is only a sufficient condition and does not guarantee Algorithm 8 will produce an optimal solution. This is only guaranteed in case there is no *duality gap* and we have $q(\lambda^*,\mu^*) = g(x,y^*)$.

For the special case of integer linear programs we have considered earlier, the above result can be strengthened [43]. In particular, let $\mathcal{D} = \{y \in \mathcal{G}\colon Ay \leq b, y \text{ is integer}\}$ be a finite subset of the integer lattice. Then, solving the Lagrangian dual problem (4.64) is identical to solving the dual of the following modified primal problem.

$$\max_y \; g(x,y) \tag{4.70}$$

$$\text{sb.t. } y \in \operatorname{conv}(\mathcal{D}), \tag{4.71}$$

$$y \in \mathcal{C}. \tag{4.72}$$

Furthermore, in the special case when we have $\operatorname{conv}(\mathcal{D}) = \{y \in \mathcal{G}\colon Ay \leq b\}$ such that $\{y \in \mathcal{G}\colon Ay \leq b\}$ is an *integral polytope*, there is no duality gap and Lagrangian relaxation is exact. In all cases, we can *recover a primal solution* y^* to (4.70) from the iterates of the subgradient method due to a result of Shor [6], that guarantees

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} y(\lambda^t,\mu^t) \to y^*. \tag{4.73}$$

In practise another popular method originally proposed by Barahona and Anbil [8] to obtain an approximate primal solution is to take the average of all primal iterates as a geometric series and thus obtaining a moving average $\bar{y}^t$ as approximate primal solution,

$$\bar{y}^t = \gamma y(\lambda^t,\mu^t) + (1-\gamma)\bar{y}^{t-1}, \tag{4.74}$$

where $0 < \gamma < 1$ is a small constant such as $\gamma = 0.1$, and $\bar{y}^t$ might be slightly primal infeasible.

Example 4.11 (MAP–MRF Message Passing). In Example 4.9 we have formulated the MAP–MRF problem as an integer linear programming problem. In this example we will apply Lagrangian relaxation to (4.48), obtaining as a byproduct a "message-passing" algorithm to solve the linear programming relaxation of (4.48). This derivation is similar to recently proposed efficient MAP inference algorithms, such as in [64, 82, 160, 162].

To do this, we first augment the original formulation with the following *implied* constraint.

$$\sum_{(y_i,y_j)\in\mathcal{Y}_i\times\mathcal{Y}_j} \mu_{i,j}(y_i,y_j) = 1, \quad \forall\{i,j\}\in E. \tag{4.75}$$

The constraint is superfluous in the original formulation because summing over (4.50) and substituting by (4.49) implies (4.75). It is, however, necessary to make it explicit. We now have the following formulation of the MAP–MRF problem.

$$\min_{\boldsymbol{\mu}} \sum_{i\in V}\sum_{y_i\in\mathcal{Y}_i} \theta_i(y_i)\mu_i(y_i) + \sum_{\{i,j\}\in E}\sum_{\substack{(y_i,y_j)\in\\ \mathcal{Y}_i\times\mathcal{Y}_j}} \theta_{i,j}(y_i,y_j)\mu_{i,j}(y_i,y_j) \tag{4.76}$$

$$\text{sb.t.} \sum_{y_j\in\mathcal{Y}_j} \mu_{i,j}(y_i,y_j) = \mu_i(y_i), \quad \forall\{i,j\}\in E, \forall y_i\in\mathcal{Y}_i \tag{4.77}$$

$$(4.49), (4.51), (4.52), (4.75).$$

Note that if constraint (4.77) were absent, then the problem could be solved trivially in linear time by setting $\mu_i(y_i) = 1$ for the $y_i \in \mathcal{Y}_i$ that minimizes the energy among all θ_i, and likewise for $\mu_{i,j}$. Therefore (4.77) is a *complicating constraint* and this is the structure Lagrangian relaxation can be beneficially applied to.

Therefore, we apply Lagrangian relaxation to the constraint (4.77) by introducing Lagrange multipliers $\lambda_{i,j}(y_i) \in \mathbb{R}$, one for each constraint (4.77). We obtain the dual function $q(\boldsymbol{\lambda})$ as the following partially dualized problem.

$$
\begin{aligned}
q(\boldsymbol{\lambda}) := \min_{\boldsymbol{\mu}} & \sum_{i \in V} \sum_{y_i \in \mathcal{Y}_i} \theta_i(y_i)\mu_i(y_i) + \sum_{\substack{\{i,j\} \\ \in E}} \sum_{\substack{(y_i,y_j) \\ \in \mathcal{Y}_i \times \mathcal{Y}_j}} \theta_{i,j}(y_i,y_j)\mu_{i,j}(y_i,y_j) \\
& + \sum_{\{i,j\} \in E} \sum_{y_i \in \mathcal{Y}_i} \lambda_{i,j}(y_i) \left(\sum_{y_j \in \mathcal{Y}_j} \mu_{i,j}(y_i,y_j) - \mu_i(y_i) \right) \qquad (4.78) \\
\text{sb.t. } & (4.49), (4.51), (4.52), (4.75).
\end{aligned}
$$

If we group the terms in the objective according to the variables, we see that for a given $\boldsymbol{\lambda}$ we can evaluate $q(\boldsymbol{\lambda})$ by separately finding the minimizing state for each variable and factor.

$$
\begin{aligned}
q(\boldsymbol{\lambda}) := \min_{\boldsymbol{\mu}} & \sum_{i \in V} \sum_{y_i \in \mathcal{Y}_i} \left(\theta_i(y_i) - \sum_{j \in V : \{i,j\} \in E} \lambda_{i,j}(y_i) \right) \mu_i(y_i) \qquad (4.79) \\
& + \sum_{\substack{\{i,j\} \\ \in E}} \sum_{\substack{(y_i,y_j) \\ \in \mathcal{Y}_i \times \mathcal{Y}_j}} \left(\theta_{i,j}(y_i,y_j) + \lambda_{i,j}(y_i) + \lambda_{j,i}(y_j) \right) \mu_{i,j}(y_i,y_j) \\
\text{sb.t. } & (4.49), (4.51), (4.52), (4.75).
\end{aligned}
$$

The optimal solution depending on $\boldsymbol{\lambda}$ is the following.

$$
\mu_i^*(y_i) = \begin{cases} 1 & \text{if } y_i = \operatorname*{argmin}_{y_i \in \mathcal{Y}_i} (\theta_i(y_i) - \displaystyle\sum_{j \in V : \{i,j\} \in E} \lambda_{i,j}(y_i)), \\ 0 & \text{otherwise.} \end{cases}
$$

$$
\mu_{i,j}^*(y_i,y_j) = \begin{cases} 1 & \text{if } (y_i,y_j) = \operatorname*{argmin}_{(y_i,y_j) \in \mathcal{Y}_i \times \mathcal{Y}_j} (\theta_{i,j}(y_i,y_j) + \lambda_{i,j}(y_i) \\ & \qquad + \lambda_{j,i}(y_j)), \\ 0 & \text{otherwise.} \end{cases}
$$

The above decomposition makes it very efficient to evaluate $q(\boldsymbol{\lambda})$. During the evaluation, a $\boldsymbol{\lambda}$-subgradient of q can be obtained efficiently using the result of Theorem 4.3. We can therefore apply the subgradient method to maximize $q(\boldsymbol{\lambda})$ iteratively. A typical behavior of the objective is shown in Figure 4.33. The above simple method is similar

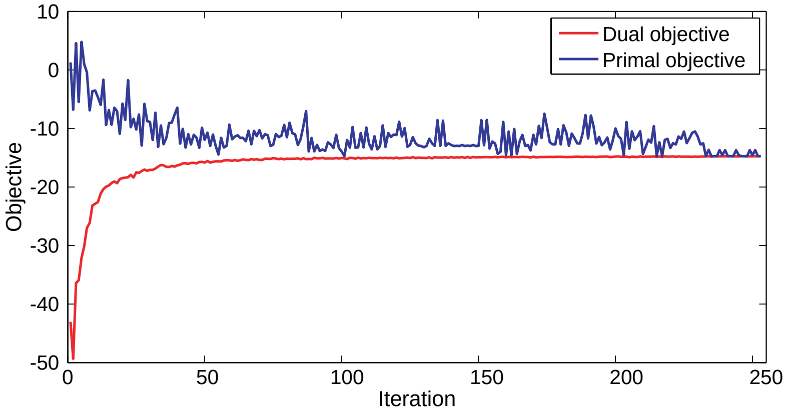


Fig. 4.33 Subgradient iterations when maximizing $q(\boldsymbol{\lambda})$ for a small submodular grid-structured MRF problem. The primal objective evaluated on $\mu^*(\boldsymbol{\lambda})$ always upper bounds the dual objective $q(\boldsymbol{\lambda})$ and in every iteration but the last the solution $\mu^*(\boldsymbol{\lambda})$ violates the dualized constraint (4.77). In this case there is no duality gap and eventually the primal and dual objectives are equal, proving optimality of the achieved objective value. A few more subgradient iterations are needed to prove solution optimality through Theorem 4.4.

in structure to the max-sum diffusion method reviewed in [162] and to max-product belief propagation. By using a subgradient method the above method always converges to an optimal dual solution $\boldsymbol{\lambda}^*$ that provides the exact MAP–MRF linear programming solution value $q(\boldsymbol{\lambda}^*)$.

4.7.3 Lagrangian Decomposition, Dual Decomposition

Lagrangian relaxation is a versatile and powerful tool to derive relaxations whenever the problem has a subset of constraints that prevent applying an efficient solution method. With Lagrangian relaxation these constraints can be treated implicitly.

If the problem does not contain such a complicating constraint set, then it turns out Lagrangian relaxation may still be applied if there is an *additive structure* as follows. Consider a special case of the prediction problem where the objective is a sum of a set of functions,

$$\begin{aligned} &\max_{y} \sum_{k=1}^{K} g_k(x,y) \\ &\text{sb.t. } y \in \mathcal{Y}_k, \quad \forall k = 1,\dots,K, \end{aligned} \tag{4.80}$$

where problem (4.80) is hard to solve, but in which the maximization of a single function is feasible, i.e., we could solve or approximate for any $k = 1, \dots, K$ the problem

$$\max_{y} \; g_k(x, y) \tag{4.81}$$

$$\text{sb.t. } y \in \mathcal{Y}_k. \tag{4.82}$$

As an example, each (4.81) could be a specially structured problem for which efficient algorithms exist. Then, we cannot directly apply these algorithms to jointly maximize (4.80).

Lagrangian decomposition allows one to solve a relaxation of (4.80) by means of iteratively solving problems of the form (4.81). The basic idea is as follows. We introduce duplicate variables y_k into the problem but add equality constraints $y = y_k$, transforming (4.80) into the following equivalent problem.

$$\max_{y, y_1, \dots, y_K} \; \sum_{k=1}^{K} g_k(x, y_k) \tag{4.83}$$

$$\text{sb.t.} \quad y_k = y, \quad \forall k = 1, \dots, K, \tag{4.84}$$

$$y_k \in \mathcal{Y}_k, \quad \forall k = 1, \dots, K, \tag{4.85}$$

$$y \in \mathcal{Y}'. \tag{4.86}$$

In (4.86) the set $\mathcal{Y}'$ can be any set containing all sets $\mathcal{Y}_k$, for example the decision domain $\mathcal{Y}' = \mathcal{G}$. Clearly, the solution to problem (4.83) is the same as to (4.80).

Problem (4.83) is amenable to Lagrangian relaxation of the complicating constraints (4.84), for if these constraints would be absent the problem would decouple into separate tractable problems of the form (4.81). We therefore introduce Lagrange multiplier vectors λ_k, one for each constraint, and dualize (4.84) to obtain the following partial dual function.

$$q(\boldsymbol{\lambda}) := \max_{y, y_1, \dots, y_K} \; \sum_{k=1}^{K} g_k(x, y_k) + \sum_{k=1}^{K} \lambda_k^\top (y_k - y) \tag{4.87}$$

$$\text{sb.t.} \quad y_k \in \mathcal{Y}_k, \quad \forall k = 1, \dots, K, \tag{4.88}$$

$$y \in \mathcal{Y}'. \tag{4.89}$$

For a given $\boldsymbol{\lambda}$, the Problem (4.87) is decomposed. The optimal solution can be obtained by separately solving for each $k = 1,\ldots,K$ the problem $\max_{y_k \in \mathcal{Y}_k} g_k(x,y_k) + \lambda_k^\top y_k$. For the maximization over $y \in \mathcal{Y}'$ we first see that (4.83) remains valid for unbounded $\mathcal{Y}'$. Then, whenever $\sum_{k=1}^K \lambda_k \neq 0$, we have $q(\boldsymbol{\lambda}) = \infty$ in (4.87). The set of $\boldsymbol{\lambda}$ on which $q(\boldsymbol{\lambda})$ remains finite defines the *domain* of q, see [13]. However, when $\sum_{k=1}^K \lambda_k = 0$, then y does not influence the value of (4.87) and we do not have to optimize over y. The dual problem of minimizing q can therefore be restricted to the domain of q and this yields the following convex minimization problem.

$$\min_{\boldsymbol{\lambda}} \; q(\boldsymbol{\lambda}) \tag{4.90}$$

$$\text{sb.t.} \sum_{k=1}^{K} \lambda_k = 0. \tag{4.91}$$

This problem is a linearly constrained convex minimization problem and can be solved by means of the *projected subgradient method* that first projects the subgradient onto (4.91) and then takes a subgradient step as usual. In our case we only have a single linear equality constraint and can provide the projected subgradient directly as follows.

$$\frac{\partial}{\partial \lambda_k} q(\boldsymbol{\lambda}) \ni y_k(\boldsymbol{\lambda}) - \frac{1}{K} \sum_{\ell=1}^{K} y_\ell(\boldsymbol{\lambda}). \tag{4.92}$$

In case we initialize $\boldsymbol{\lambda}$ such that it satisfies (4.91) the subgradient method will then continue to satisfy the constraint.

Because we first transformed the original problem into an *equivalent* one and then applied Lagrangian relaxation, we retain the strong theoretical guarantees provided by the Lagrangian relaxation approach. In particular, (4.90) yields an upper bound on (4.80) and we can recognize optimality by means of the duality gap between any primal solution value and the dual value.

For the special case where $\mathcal{Y}_k$ are finite sets — as is the case for discrete models — and we have a linear cost function $g(x,y) = c(x)^\top y$, the following theorem providing a simple interpretation of Lagrangian decomposition was proven by [54].

Theorem 4.5 (Lagrangian Decomposition Primal). Let a problem of the form (4.80) be given, where $\mathcal{Y}_k$ are finite sets and $g(x,y) = c(x)^\top y$ is a linear objective function. Then the optimal solution of the Lagrangian dual (4.90) obtains the value of the following relaxed primal optimization problem.

$$\max_{y} \sum_{k=1}^{K} g_k(x,y) \tag{4.93}$$

$$\text{sb.t. } y \in \operatorname{conv}(\mathcal{Y}_k), \quad \forall\, k = 1,\dots,K. \tag{4.94}$$

Therefore, optimizing the Lagrangian decomposition dual is equivalent to optimizing the primal objective on the intersection of the convex hulls of the individual feasible sets.

Note that the Lagrangian decomposition method is still sound when the assumptions of Theorem 4.5 do not hold, it is just the primal interpretation of (4.90) that is less clear.

In the above discussion we have assumed that the subproblems are tractable. Lagrangian decomposition can be applied without this assumption if we are able to solve a *relaxation of the subproblem*. In that case, we are still guaranteed to obtain a relaxation to the original problem, but the relaxation is weaker — its optimal value yields a weaker upper bound — than if we could solve the subproblem exactly. The geometric interpretation of what happens when such weaker relaxation is used is depicted in Figure 4.34. Although the Lagrangian decomposition method is often applicable, in some problems different choices of subproblems are possible and each choice produces one decomposition.

Lagrangian decomposition is also called "dual decomposition" in the computer vision community; another name is "variable splitting." For the original paper, see [54], for an up to date textbook covering other problem decomposition techniques, see [34].

Lagrangian/Dual Decomposition in computer vision. Lagrangian decomposition has been successfully applied to a number of computer vision problems. The connectivity constrained

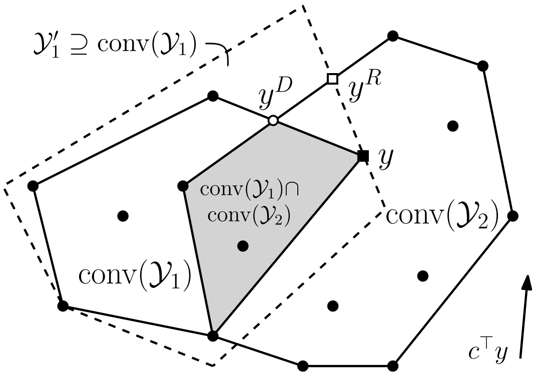

$\mathcal{Y}_1' \supseteq \text{conv}(\mathcal{Y}_1)$
y^D
y^R
y
$\text{conv}(\mathcal{Y}_1)\cap$
$\text{conv}(\mathcal{Y}_2)$
$\text{conv}(\mathcal{Y}_2)$
$\text{conv}(\mathcal{Y}_1)$
$c^\top y$

Fig. 4.34 Schematic interpretation of the geometry of Lagrangian decomposition, following [54]. Here we have two subproblems over finite sets $\mathcal{Y}_1$ and $\mathcal{Y}_2$, respectively. The cost function $g(x,y) = c^\top y$ is linear and the optimal solution to $\max_{y \in \mathcal{Y}_1 \cap \mathcal{Y}_2} c^\top y$ is y^* (marked with a square). In case each subproblem can be solved exactly, Lagrangian decomposition produces the solution y^D, that is, the maximizing solution within $\text{conv}(\mathcal{Y}_1) \cap \text{conv}(\mathcal{Y}_2)$. We therefore obtain a relaxation, i.e., we always have $c^\top y^D \geq c^\top y^*$. In case exact optimization over $\mathcal{Y}_1$ is intractable and we use a relaxation $\mathcal{Y}_1' \supseteq \text{conv}(\mathcal{Y}_1)$, then this weakens the relaxation and we obtain the solution y^R which satisfies $c^\top y^R \geq c^\top y^D \geq c^\top y^*$.

segmentation problem has been used by Vicente et al. [157] in a dual decomposition approach to obtain strong lower bounds on an objective; the decomposition approach turned out to be slow and a fast primal heuristic is used instead. Feature matching between images has been addressed using graph matching in [150], where dual decomposition provided strong relaxations. The following example is a typical situation in which the decomposition can be applied.

Example 4.11 (Segmentation with Global Appearance Models). In [158] the authors derive an approach to binary image segmentation incorporating a global appearance model of the object to be segmented. The model is similar to the one in Example 4.2 but the function $g(x,y)$ to be maximized incorporates a term $h_k(\cdot)$ that takes as argument the number of pixels labeled as foreground. Because this function is nonlinear it does not decompose and finding the maximizer of $g(x,y)$ is NP-hard.

For the simplest model in [158] with binary label set $\mathcal{L} = \{0,1\}$, the function $g(x,y)$ is given as follows.

$$g(x,y) = \underbrace{-\sum_{b\in B} h_b\left(\sum_{i\in V}[\![i \in V_b]\!]\right) - \sum_{(i,j)\in\mathcal{E}} w_{i,j}[\![y_i \neq y_j]\!]}_{g_1(x,y)}\underbrace{-h\left(\sum_{i\in V}[\![y_i = 1]\!]\right)}_{g_2(x,y)}, \tag{4.95}$$

where $h_b(\cdot)$ is a concave function and $h(\cdot)$ is a convex function. These functions are derived from a global histogram-based color model. In this model, a set B of fixed histogram bins *a priori* partitions the pixel set V, i.e., we have $\bigcup_{b\in B} V_b = V$. The set $V_b \subset V$ contains the set of nodes that have a pixel color mapped to the histogram bin b. The functions h_b and h make the model prefer labelings whose foreground and background pixels are forming a consistent color model. For details on these models, see [158]. Optimizing (4.95) is NP-hard but when suitably decomposed a relaxation can be optimized efficiently.

In order to apply the Lagrangian decomposition scheme to the two parts g_1 and g_2 we split the variable y into y_1 and y_2 and obtain the two problems,

- $\max_{y_1\in\mathcal{Y}} [g_1(x,y_1) + \lambda_1^\top(y_1 - y)]$, and
- $\max_{y_2\in\mathcal{Y}} [g_2(x,y_2) + \lambda_2^\top(y_2 - y)]$.

Both subproblems can be solved in polynomial time: the first problem is known to be solvable by reducing it to a linear $s - t$ min-cut problem, as shown by Kohli et al. [76], but Vicente et al. [158] propose a more efficient procedure. The second subproblem has a simple structure and can also be solved efficiently. Applying the subgradient method to minimize the dual problem solves a relaxation and provides lower and upper bounds on the optimal objective. The authors empirically observe that in most cases the relaxation is in fact integral and provides the optimal solution y^*. An example segmentation using a global color model is shown in Figures 4.35 to 4.37.

The above ideas have been generalized to the so called *marginal probability field* model in [169].

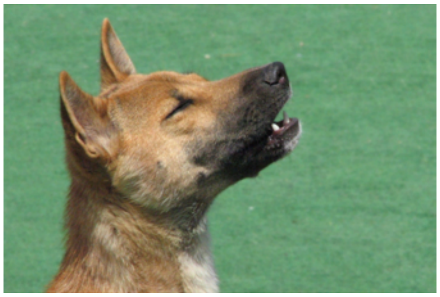


Fig. 4.35 Input image to be segmented.

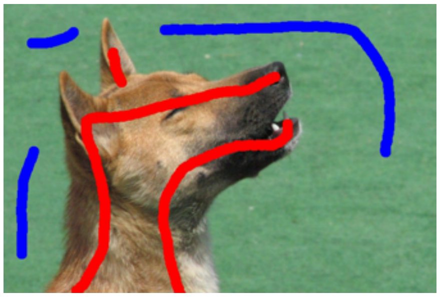


Fig. 4.36 User annotation of foreground and background.

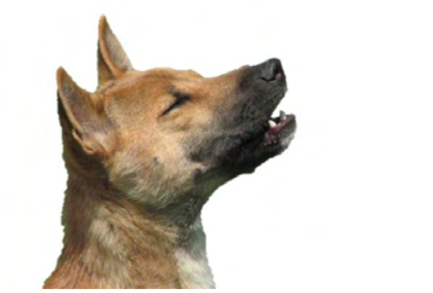


Fig. 4.37 Final segmented image.

The Lagrangian decomposition method was applied to the MAP–MRF problem by Johnson et al. [64] and Komodakis et al. [82] to approximately solve the linear programming relaxation by iterating MAP inference on tree-structured subgraphs. The family of possible decompositions for MAP inference in graphical models has been further studied by Werner [164], Franc et al. [42] and efficiently solvable higher-order decompositions yielding stronger bounds are proposed in [11]. Sontag et al. [141] provides a detailed introduction to dual decomposition and dual ascent methods for inference.

Decomposing the original problem into smaller instances naturally allows for parallel and distributed computation. This is used by Strandmark and Kahl [143] in a straightforward approach to decompose large discrete MAP inference problems with more than 600 million voxels onto multiple machines, each solving only a smaller subproblem.

A number of recent works have improved on the subgradient method used to optimize the Lagrangian decomposition dual (4.90). Jojic et al. [65] and also Savchynskyy et al. [130] applied Nesterov's smoothing technique [110] to (4.90). This improves the asymptotic convergence rate at the cost of modifying the subproblem (4.87) by the addition of a strictly concave proximal regularization term. For the case of discrete graphical models the resulting subproblem can still be solved by replacing the energy minimization step with probabilistic inference. In the same spirit, but from the perspective of augmented Lagrangian optimization, Martins et al. [100] augment the subproblems by a strictly convex proximal term stabilizing the iterations. This complicates the subproblems slightly but leads to a large decrease in the number of outer iterations.

4.7.4 General MAP–MRF Linear Programming Relaxations

The pairwise LP relaxation (4.48), also known as the *LOCAL relaxation* [160], was initially proposed in Schlesinger [131] but has since been extended to models with factors of higher-order. In the following we describe the most general known family of cluster-based relaxations, as proposed by Werner [163] and described in detail in Franc et al. [42]. This family of relaxations is so large that it contains both the LOCAL relaxation and the exact marginal polytope as special cases. As such, not all relaxations in this family are tractable.

The construction of this family has close ties to *region graphs* and free energy approximations used for probabilistic inference, see Yedidia et al. [171] and Heskes [59]. The basic idea of enforcing marginalization consistency over larger groups of variables can be dated back to Kikuchi [69].

For a given factor graph $(V,\mathcal{F},\mathcal{E})$, each relaxation instance is defined by means of a set of triplets, $J = \{(A,B,y_B) | A,B \subseteq V, \emptyset \neq B \subset A\}$, the so called *pencil set* [163]. For each factor $F \in \mathcal{F}$ present in the model, there must exist at least one pencil $(A,\cdot,\cdot)$, where $A = N(F)$ is the set of variables in the scope of the factor. The construction is visualized for higher-order factors in Figure 4.38. For brevity, we write $A \in J$ if there exists one element $(A,B,y_B) \in J$ or one element $(C,A,y_A) \in J$. For any such set J we can now obtain a valid relaxation $\mathcal{L}(J)$ of the MAP–MRF problem by formulating a linear program as shown in Figure 4.39. The coefficients $\theta_A(y_A)$ are defined as

$$\theta_A(y_A) = \sum_{F \in \mathcal{F}: N(F) = A} E_F(y_A), \tag{4.96}$$

and we can have $\theta_A(y_A) = 0$ if the set A does not correspond to the scope of any factor in the model.

For example, we can obtain the LOCAL relaxation for a factor graph with pairwise factors by setting

$$J_{\text{LOCAL}} = \{(A,B,x_B) | F \in \mathcal{F}, |N(F)| = 2, A = N(F), B \subset A, y_B \in \mathcal{Y}_B\}.$$

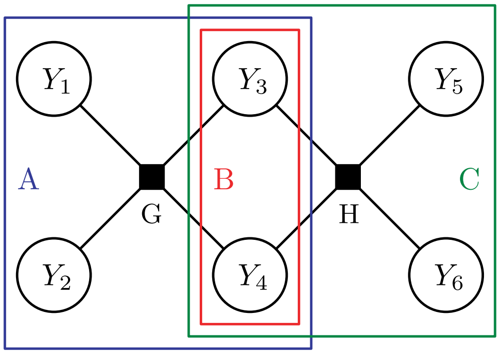


Fig. 4.38 Constructing a linear programming relaxation for higher-order factors. The model contains two factors, that is, $\mathcal{F} = \{G,H\}$. We define two sets of pencils by (A,B,x_B), and (C,B,x_B), where $A = N(G) = \{Y_1,Y_2,Y_3,Y_4\}$, $B = N(G) \cap N(H) = \{Y_3,Y_4\}$, and $C = N(H) = \{Y_3,Y_4,Y_5,Y_6\}$. This introduces the marginal vector μ_B that ensures the marginal vectors μ_A and μ_C are consistent with each other when marginalized onto B, their common set of variables.

$$
\begin{aligned}
\min_{\boldsymbol{\mu}} \quad & \langle \boldsymbol{\theta}, \boldsymbol{\mu} \rangle && && (4.97)\\
\text{sb.t.} \quad & \sum_{y_A} \mu_A(y_A) = 1, && \forall A \in J, && (4.98)\\
& \sum_{y_{A \setminus B}} \mu_A(y_A) = \mu_B(y_B), && \forall (A, B, y_B) \in J, && (4.99)\\
& \mu_A(y_A) \geq 0, && \forall A \in J, y_A. && (4.100)
\end{aligned}
$$

Fig. 4.39 Primal linear program of the cluster-based relaxation $\mathcal{L}(J)$.

In this case, (4.97–4.100) reduces to the earlier linear relaxation (4.48) of Schlesinger [131]. If instead we take the largest possible set

$$J_{\text{ALL}} = \{(A, B, y_B) | A \in 2^V, B \subset A, y_B \in \mathcal{Y}_B\},$$

then for each factor $F \in \mathcal{F}$ the corresponding marginals $\mu_{N(F)}$ are globally consistent due to (4.99), and the solution of (4.97) is the exact minimum energy solution. Unfortunately, J_{ALL} grows exponentially in the model size and is therefore impractical.

Therefore, one typically starts with the LOCAL relaxation and iteratively enforces consistency over larger groups of variables. For any given particular relaxation, most algorithms solve (4.97) by means of the *dual* linear program (4.101). The reason is that ensuring dual feasibility for (4.104) is straightforward. The algorithms of Sontag et al. [140, 142], Werner [163], and Komodakis and Paragios [81] are all based on the dual optimization problem (4.101).

The selection of groups of variables and pencils to use in a relaxation remains an open issue. For a discussion of the problems involved, see Franc et al. [42] and Bara et al. [12].

4.8 Giving up Determinism

Giving up the determinism in computation leads to algorithms whose output are random variables. Alternatively, the algorithm might give a deterministic result but its runtime is random.

$$\max_{\boldsymbol{h},\boldsymbol{z}} \quad \sum_{A \in J} h_A \tag{4.101}$$

$$\text{sb.t.} \quad h_A \in \mathbb{R}, \qquad \forall A \in J, \tag{4.102}$$

$$z_{A \to B}(y_B) \in \mathbb{R}, \qquad \forall (A,B,y_B) \in J, \tag{4.103}$$

$$\begin{aligned} h_A \leq \theta_A(y_A) &+ \sum_{Z|(Z,A,y_A) \in J} z_{Z \to A}(y_A) \\ &- \sum_{B|(A,B,y_B) \in J} z_{A \to B}(y_B), \qquad \forall A \in J, y_A. \end{aligned} \tag{4.104}$$

Fig. 4.40 Dual linear program of the cluster-based relaxation $\mathcal{L}(J)$.

We have already seen the usefulness of sampling techniques in *Monte Carlo*, now we consider solving optimization problems using randomized algorithms.

4.8.1 Simulated Annealing

Simulated annealing [1, 71] is a well-known method to approximately solve problems of the form (4.2). It consists of two steps, (i) constructing a family of probability distributions $p(y;T)$ on $\mathcal{Y}$, and (ii) a method to approximately generating samples from these distributions.

For the first step, the distributions $p(y;T)$ are constructed in such a way that most of the probability mass is concentrated on states $y \in \mathcal{Y}$ with high values $g(x,y)$. The distribution $p(y;T)$ is constructed as a *Boltzmann distribution*, defined as follows.

Definition 4.3 (Boltzmann Distribution). For a finite set $\mathcal{Y}$, a function $g\colon \mathcal{Y} \to \mathbb{R}$ and a *temperature parameter* $\tau > 0$, let

$$p(y;\tau) = \frac{1}{Z(\tau)} \exp\left(\frac{g(x,y)}{\tau}\right), \tag{4.105}$$

with $Z(\tau) = \sum_{y \in \mathcal{Y}} \exp(\frac{g(x,y)}{\tau})$ be the *Boltzmann distribution* for g over $\mathcal{Y}$ at temperature τ. The Boltzmann distribution is also known as the *Gibbs measure* or *Gibbs distribution.*

Figures 4.41 to 4.46 illustrate the behavior of the Boltzmann distribution with different values of τ for a simple case where $\mathcal{Y} = \{1, 2, \ldots, 40\}$. The smaller the temperature becomes, the more probability mass is put on the state y^* that attains the maximum value $g(x, y^*)$, and we have $\lim_{\tau \to 0} p(y^*; \tau) = 1$.

For this example, finding the maximizer is of course trivial by enumerating all elements in $\mathcal{Y}$. In general we are interested in problems where $\mathcal{Y}$ is large and therefore such enumeration is intractable. Likewise, naively drawing samples from $p(y; \tau)$ by computing (4.105) for all $y \in \mathcal{Y}$ is intractable. However, drawing approximate samples is possible by constructing *Markov chain* on $\mathcal{Y}$ as discussed in *Inference in Graphical Models*. Drawing a sample at a low temperature makes it more likely that the obtained iterate is the maximizer of g. Simulated annealing achieves this by starting with a high temperature and lowering it gradually.

Before we show how a Markov chain simulating $p(y; \tau)$ can be constructed, we first discuss the simulated annealing algorithm, as shown in Algorithm 9. The algorithm takes a sequence $\tau^{(1)}, \tau^{(2)}, \ldots, \tau^{(K)}$ of monotonically decreasing temperatures and a sequence $\eta^{(1)}, \eta^{(2)}, \ldots, \eta^{(K)}$ of

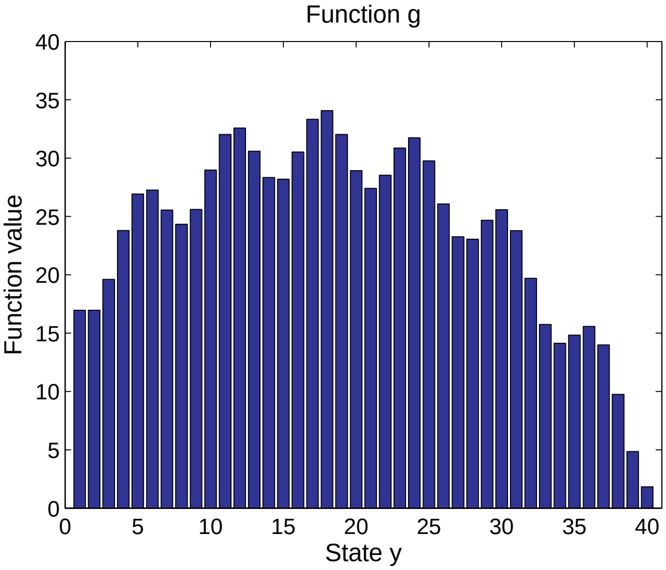


Fig. 4.41 Function g to be maximized.

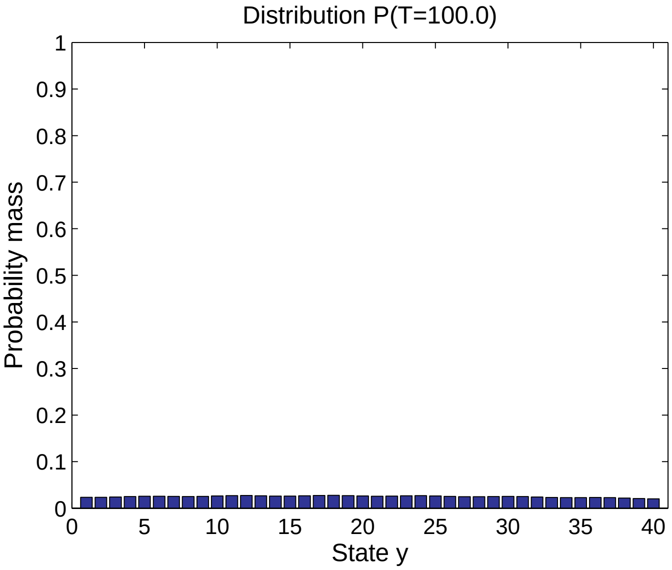


Fig. 4.42 Distribution $p(y; \tau)$ for $\tau = 100$.

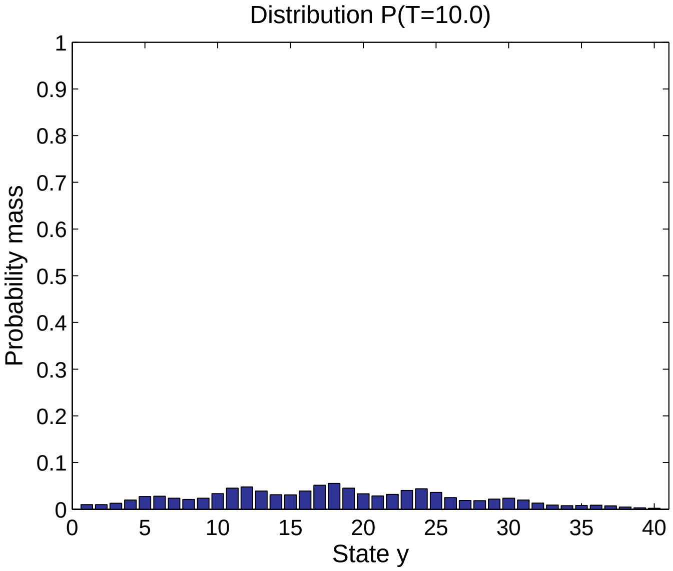


Fig. 4.43 Distribution $p(y; \tau)$ for $\tau = 10$.

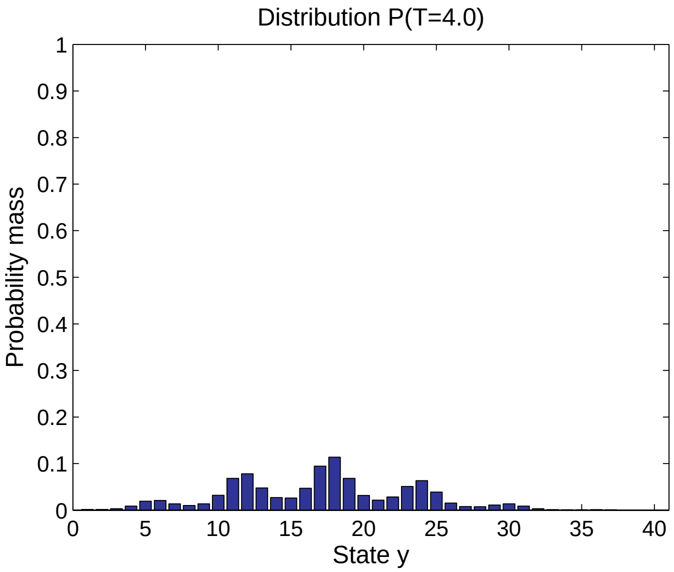


Fig. 4.44 Distribution $p(y;\tau)$ for $\tau = 4$.

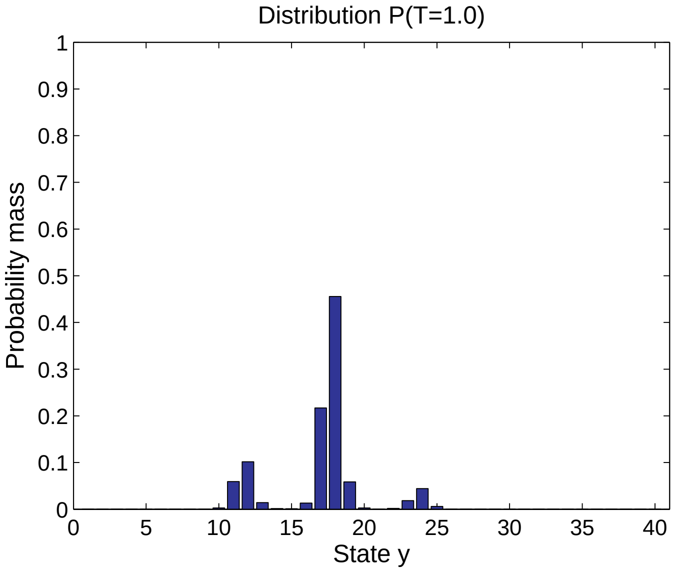


Fig. 4.45 Distribution $p(y;\tau)$ for $\tau = 1$.

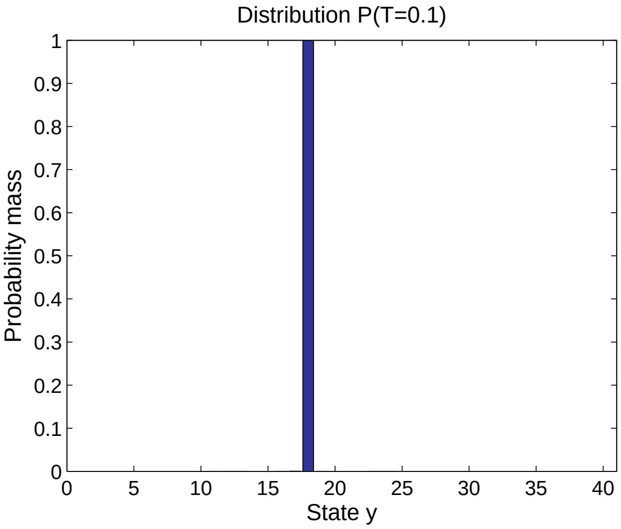


Fig. 4.46 Distribution $p(y;\tau)$ for $\tau = 0.1$.

steps to simulate at each temperature. For each temperature $\tau^{(k)}$ a Markov chain is simulated for the given number of steps $\eta^{(k)}$, hopefully improving on the solution objective. Any Markov chain with $p(y;\tau^{(k)})$ as stationary distribution can be used, such as the Metropolis–Hastings chain or the Gibbs sampler discussed before. Throughout the algorithm the best solution observed so far is kept.

Finally, the choice of temperatures $\tau^{(k)}$ and number of simulations $\eta^{(k)}$ over time, the so called *annealing schedule* must be set. In case the temperature decreases slowly enough, the following theoretical result is known due to Geman and Geman [49].

Theorem 4.6 (Guaranteed Optimality Annealing). If there exist a $k_0 \geq 2$ such that for all $k \geq k_0$ the temperature $\tau^{(k)}$ satisfies the lower bound

$$\tau^{(k)} \geq \frac{|\mathcal{Y}| \cdot (\max_{y \in \mathcal{Y}} g(x,y) - \min_{y \in \mathcal{Y}} g(x,y))}{\log k}, \tag{4.106}$$

then the probability of seeing the maximizer y^* of g tends to one as $k \to \infty$. For the original proof, see [49].

Algorithm 9: Simulated Annealing

```
$y^* = \textsc{SimulatedAnnealing}(\mathcal{Y}, g, \tau, N)$
Input:
    $\mathcal{Y}$ finite feasible set
    $g : \mathcal{Y} \to \mathbb{R}$ objective function
    $\tau \in \mathbb{R}^K$ sequence of $K$ decreasing temperatures
    $\eta \in \mathbb{N}^K$ sequence of $K$ step lengths
Output:
    $y^* \in \mathcal{Y}$ solution
Algorithm:
$y \leftarrow y_0$
$y^* \leftarrow y_0$
for $k = 1, \ldots, K$ do
    for $i = 1, \ldots, \eta^{(k)}$ do
        $y \leftarrow$ run Markov chain simulating $p(y; \tau^{(k)})$ for one step
        if $g(x,y) > g(x,y^*)$ then
            $y^* \leftarrow y$ {Improved solution}
        end if
    end for
end for
```

The above bound (4.106) goes to zero too slowly to be useful in practise. Instead, the so called *exponential schedule*

$$\tau^{(k)} = \tau^{(0)} \cdot \alpha^k, \tag{4.107}$$

is popular, where $\tau^{(0)} > 0$ is a fixed constant and α is set to a number close to one, e.g., $\alpha = 0.999$. The simulation intervals are often fixed such as setting $\eta^{(k)} = 10$ for all k.

Example 4.13 (Image Restoration). The influential paper of Geman and Geman [49] proposed the *Gibbs sampler* for obtaining approximate samples from otherwise intractable distributions. The authors also used the newly proposed sampler to perform approximate MAP inference by simulated annealing. The following experiment is an exact replication of the first experiment in that paper.

We assume a 128 × 128 pixel image with each pixel taking one out of five possible intensity values. We define the Gibbs distribution

$$p(y) = \frac{1}{Z}\exp(-E(y)), \tag{4.108}$$

where the energy is a sum of pairwise terms, i.e., $E(y) = \sum_{(i,j)\in\mathcal{E}} V_{i,j}(y_i, y_j)$, with $V_{i,j}$ taking the Potts form:[6]

$$V_{i,j}(y_i, y_j) = \begin{cases} -\frac{1}{3} & \text{if } y_i = y_j, \\ \frac{1}{3} & \text{otherwise.} \end{cases}$$

There are no unary factors. We obtain an approximate sample y^* from the distribution by Gibbs sampling, the result of which is shown in Figure 4.47. The task is to denoise a noisy version of this image.

To this end, we use independent additive Gaussian noise to obtain the noisy image y' shown in Figure 4.48. Geman and Geman show that the posterior distribution $p(y|y_{\text{noisy}})$ is again of the form (4.108) with

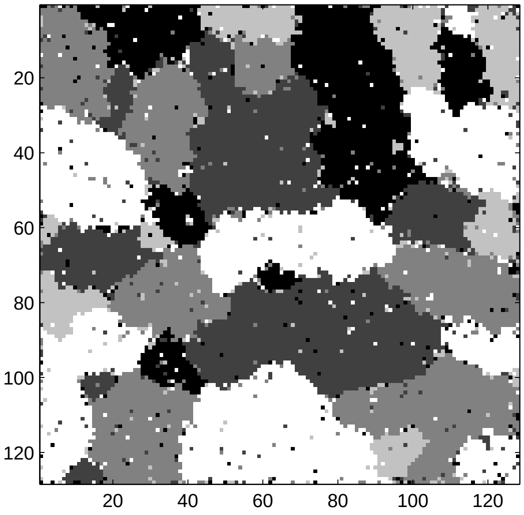


Fig. 4.47 Sample y^* from prior distribution with five labels, obtained using 200 Gibbs sampler sweeps.

[6] The original form given in [49, Page 12] contains a sign error.

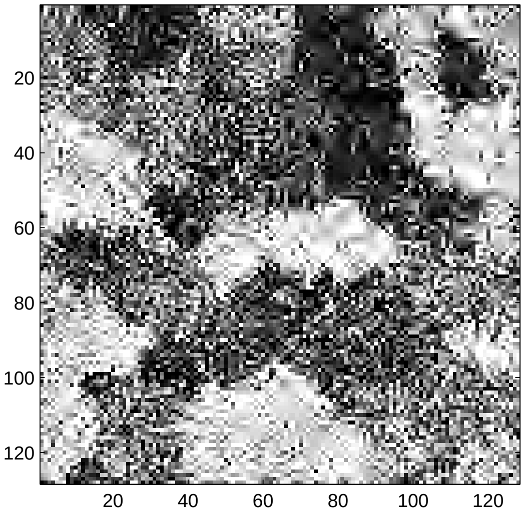


Fig. 4.48 Sample y' with independent additive Gaussian noise, $\sigma = 1.5$.

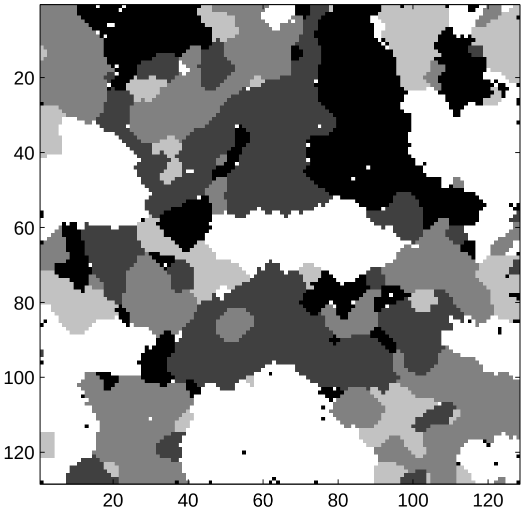


Fig. 4.49 Reconstruction with 25 simulated annealing sweeps.

the modified energy $E^P(y)$ as

$$E^P(y) = E(y) + \frac{1}{2\sigma^2} \sum_i (y_i' - y_i)^2. \tag{4.109}$$

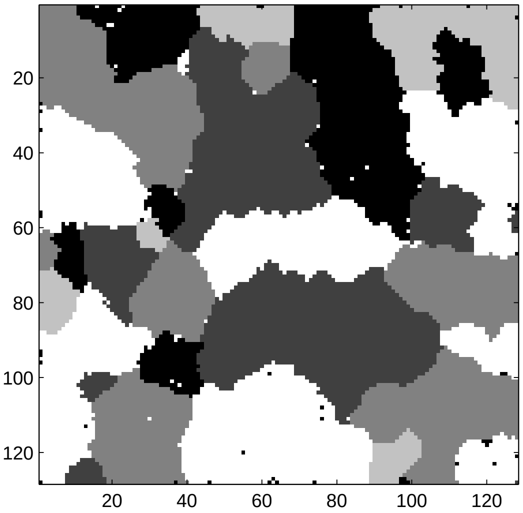


Fig. 4.50 Reconstruction with 300 simulated annealing sweeps.

This simply adds unary factors to the previous model and we can minimize $E^P(y)$ by simulated annealing, using a Gibbs sampler to sample from the intermediate distributions. The results of two different runs using 25 and 300 simulated annealing sweeps is shown in Figures 4.49 and 4.50, respectively.

Simulated annealing can be particularly effective if a problem-specific sampler is used. For example, in [10] image segmentation is performed by simulated annealing, but the simulation is carried out using an efficient sampler for partitionings, giving rise to a hundred-fold speedup over naive single-variable Gibbs sampling. A good general reference on simulated annealing is [1].

5

Conditional Random Fields

We now turn to the problem of *probabilistic parameter learning*. For this we assume a fixed underlying graphical model with parameterized conditional probability distribution

$$p(y|x,w) = \frac{1}{Z(x,w)} \exp(-E(x,y,w)), \tag{5.1}$$

where $Z(x,w) = \sum_{y \in \mathcal{Y}} \exp(-E(x,y,w))$. The only unknown quantity is the *parameter* or *weight vector* w, on which the energy $E(x,y,w) = \langle w, \varphi(x,y) \rangle$ depends linearly.

As introduced in *Problem 2.3* of *Graphical Models*, probabilistic learning aims at identifying a weight vector w^* that makes $p(y|x,w^*)$ as close to the true conditional label distribution $d(y|x)$ as possible. The label distribution itself is unknown to us, but we have an *i.i.d.* sample set $\mathcal{D} = \{(x^n,y^n)\}_{n=1,\dots,N}$ from $d(x,y)$ that we can use for learning.

5.1 Maximizing the Conditional Likelihood

The most common principle for probabilistic training is *(regularized) conditional likelihood maximization* that we define and discuss in this section. Training a graphical model in this way is generally called *conditional random field (CRF)* training.

Definition 5.1 (Regularized Maximum Conditional Likelihood Training). Let $p(y|x,w) = \frac{1}{Z(x,w)} \exp(-\langle w, \varphi(x,y)\rangle)$ be a probability distribution parameterized by $w \in \mathbb{R}^D$, and let $\mathcal{D} = \{(x^n, y^n)\}_{n=1,\ldots,N}$ be a set of training examples. For any $\lambda > 0$, *regularized maximum conditional likelihood (RMCL)* training chooses the parameter as

$$w^* = \operatorname{argmin}_{w\in\mathbb{R}^D} \;\; \lambda\|w\|^2 + \sum_{i=n}^{N} \langle w, \varphi(x^n, y^n)\rangle + \sum_{n=1}^{N} \log Z(x^n, w). \tag{5.2}$$

For $\lambda = 0$ the simplified rule

$$w^* = \operatorname{argmin}_{w\in\mathbb{R}^D} \;\; \sum_{n=1}^{N} \langle w, \varphi(x^n, y^n)\rangle + \sum_{n=1}^{N} \log Z(x^n, w) \tag{5.3}$$

is called *maximum conditional likelihood (MCL)* training.

To understand the objective behind Equations (5.2) and (5.3) let us try to solve Problem 2.3 in a straight-forward way, i.e., derive a weight vector w^* that makes $p(y|x,w^*)$ closest to $d(y|x)$. First we define what we mean by "closeness" between conditional distributions: for any $x \in \mathcal{X}$, we measure the dissimilarity between $p(y|x,w)$ and $d(y|x)$ by their Kullback–Leibler (KL) divergence: $\mathrm{KL}(p\|d) = \sum_{y\in\mathcal{Y}} d(y|x) \log \frac{d(y|x)}{p(y|x,w)}$. From this we obtain a total measure of how much q differs from d by their expected dissimilarity over all $x \in \mathcal{X}$:

$$\mathrm{KL}_{\mathrm{tot}}(p\|d) = \sum_{x\in\mathcal{X}} d(x) \sum_{y\in\mathcal{Y}} d(y|x) \log \frac{d(y|x)}{p(y|x,w)}. \tag{5.4}$$

We solve Problem 2.3 by choosing the parameter w^* that minimizes this measure, i.e.,

$$\begin{aligned} w^* &= \operatorname{argmin}_{w\in\mathbb{R}^d} \;\; \mathrm{KL}_{\mathrm{tot}}(p\|d) & (5.5)\\ &= \operatorname*{argmax}_{w\in\mathbb{R}^d} \sum_{x\in\mathcal{X}} d(x) \sum_{y\in\mathcal{Y}} d(y|x) \log p(y|x,w) & (5.6)\\ &= \operatorname*{argmax}_{w\in\mathbb{R}^d} \;\; \mathbb{E}_{(x,y)\sim d(x,y)}[\log p(y|x,w)]. & (5.7) \end{aligned}$$

Unfortunately, we cannot compute this expression directly, because $d(x,y)$ is unknown to us. However, we can approximate it using the given sample set $\mathcal{D}$.

$$\approx \operatorname*{argmax}_{w\in\mathbb{R}^d} \sum_{(x^n,y^n)\in\mathcal{D}} \log p(y^n|x^n,w). \tag{5.8}$$

Inserting the parametric form of p, we see that Equation (5.8) is equivalent to Equation (5.3). Making use of $p(y^1,\dots,y^N|x^1,\dots,x^N,w) = \prod_{n=1}^N p(y^n|x^n,w)$, because $\mathcal{D}$ is sampled *i.i.d.*, and the monotonicity of the logarithm, we obtain

$$= \operatorname*{argmax}_{w\in\mathbb{R}^d} p(y^1,\dots,y^N|x^1,\dots,x^N,w). \tag{5.9}$$

from which the name MCL stems.

MCL training has been applied successfully to many learning tasks where the dimensionality of the weight vector is small and the number of training examples is large. However, when the number of training instances is small compared to the number of degrees of freedom in w then MCL training is prone to overfitting. An explanation is that the approximation leading to Equation (5.8) becomes unreliable in this situation, and w^* will vary strongly with respect to the training set $\mathcal{D}$.

It is possible to overcome this limitation by treating w not as a deterministic parameter but as yet another random variable.[1] For any prior distribution $p(w)$ over the space of weight vectors, one can derive the posterior probability of w for given observations $\mathcal{D}$:

$$\begin{aligned} p(w|\mathcal{D}) &= \frac{p(w)p(y^1,\dots,y^N|x^1,\dots,x^N,w)}{p(y^1,\dots,y^N|x^1,\dots,x^N)}, \\ &= p(w)\prod_{n=1}^N \frac{p(y^n|x^n,w)}{p(y^n|x^n)}, \end{aligned} \tag{5.10}$$

where in the first step we have made use of Bayes' rule, and in the second step of the *i.i.d.* assumption on $\mathcal{D}$. Having to choose a single

[1] This view makes a lot of sense, since our choice of w^* depends on $\mathcal{D}$, which itself is randomly sampled.

weight vector it makes sense to use the *maximum a posteriori* estimate,

$$w^* = \operatorname*{argmax}_{w \in \mathbb{R}^d} \; p(w|\mathcal{D}) \tag{5.11}$$

$$= \operatorname*{argmin}_{w} \left[-\log p(w) - \sum_{n=1}^{N} \log p(y^n|x^n, w) \right]. \tag{5.12}$$

The second term is the negative logarithm of the conditional likelihood, which we have already encountered in MCL training. In the first term, $p(w)$ expressed our prior assumptions on w. We have to specify it as a design choice, because as a prior it cannot be estimated from data. Assuming a zero-mean Gaussian prior,[2] $p(w) \propto \exp(-\frac{\|w\|^2}{2\sigma^2})$, the first term in Equation (5.12) becomes $\frac{1}{2\sigma^2}\|w\|^2$. In combination we recover Equation (5.2) with $\lambda = \frac{1}{2\sigma^2}$.

The name *RMCL* reflects that for $\lambda > 0$ very large positive and very large negative values in the weight vector are penalized by the prior term. Consequently, the parameter λ is generally considered as a free hyper-parameter that determines the *regularization strength.*

In the remainder of the section, we will only study regularized conditional likelihood maximization, as the unregularized situation can be seen as the limit case for $\lambda \to 0$.

5.2 Gradient Based Optimization

Writing the negative of the logarithm of the regularized conditional likelihood (called *negative log-likelihood*) as a function of w,

$$\mathcal{L}(w) = \lambda\|w\|^2 + \sum_{n=1}^{N} \langle w, \varphi(x^n, y^n) \rangle + \sum_{n=1}^{N} \log Z(x^n, w), \tag{5.13}$$

one sees that the first two terms are relatively simple, with only a quadratic and linear dependence on w. The third term, however, contains the partition function $Z(x,w)$ which depends in a nonlinear and nonobvious way on the typically high-dimensional vector of unknowns w. Because of this term the optimization problem (5.2) does

[2] Other priors are possible but less common. See, e.g., [52] for an overview of exponential family priors.

not have a closed form solution, and we need to rely on numerical optimization techniques to find its minimum. Figure 5.1 illustrates the contour lines of $\mathcal{L}(w)$ for different values of λ.

High dimensional nonlinear optimization can be difficult, but $\mathcal{L}(w)$ is a *smooth convex function* as we can see from the Jacobian vector $\nabla_w \mathcal{L}(w)$ and Hessian matrix $\Delta_w \mathcal{L}(w)$:

$$\nabla_w \mathcal{L}(w) = 2\lambda w + \sum_{n=1}^{N} [\varphi(x^n, y^n) - \mathbb{E}_{y \sim p(y|x^n)}[\varphi(x^n, y)]] \tag{5.14}$$

and

$$H_w \mathcal{L}(w) = 2\lambda I + \sum_{n=1}^{N} \mathbb{E}_{y \sim p(y|x^n)} \mathbb{E}_{y' \sim p(y|x^n)} [\varphi(x^n, y) \varphi^t(x^n, y')]. \tag{5.15}$$

Because the last term is a covariance matrix it is positive semi-definite and this proves the convexity of $\mathcal{L}$. For nonzero λ, $H_w \mathcal{L}$ is even strictly positive definite, which ensures strong convexity of $\mathcal{L}$. Because the convexity of $\mathcal{L}$ guarantees that every local maximum will also be a global maximum, we can use local optimization techniques to minimize $\mathcal{L}$. In particular, because $\mathcal{L}$ is differentiable, we can use *gradient descent*.

5.3 Numeric Optimization

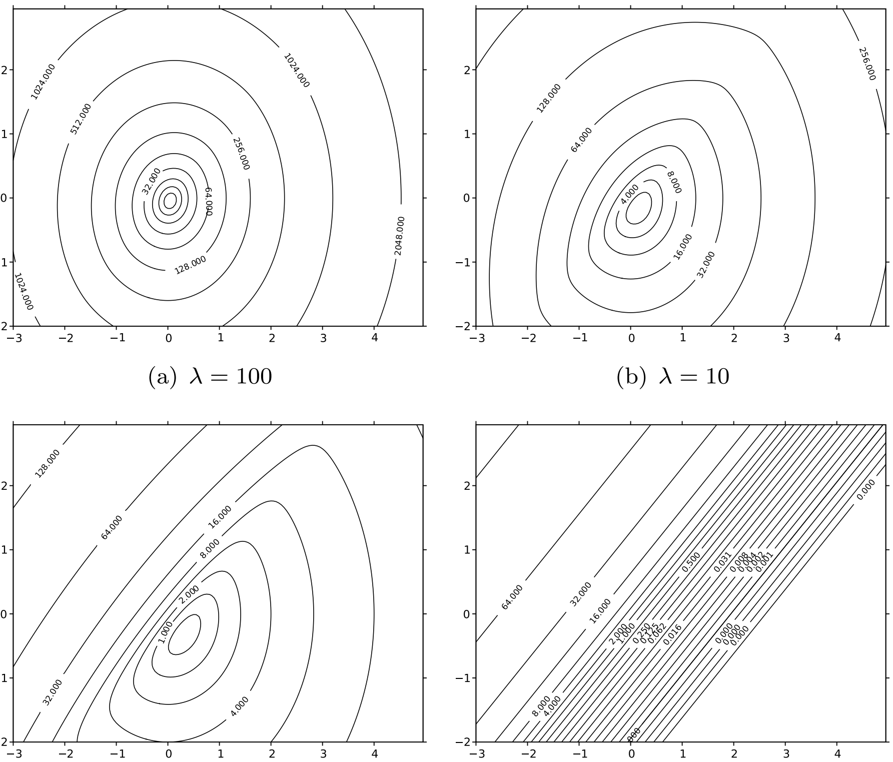
Steepest Descent Optimization. The most straight-forward technique to numerically solve the optimization problem (5.3) is the *steepest descent* algorithm for which pseudo-code is shown in Algorithm 10. Starting with an arbitrary estimate of the weight vector, typically $w = 0$ (line 7), one iteratively approaches the minimum. In each step, one computes a descent direction, which is the negative of the gradient $\nabla_w \mathcal{L}$ at the current estimate w_{cur} (line 9). The next estimate for the weight vector is obtained by adding a multiple of the negative gradient to w_{cur} (line 11). Using a line search the step width that leads to the strongest decrease of the objective function is determined (line 10). These steps are repeated until a convergence criterion is met, e.g., the magnitude of the gradient is below a given threshold (line 12), indicating that we have reached, approximately, a local minimum. Because the objective function is convex, we know that this local minimum is automatically the global minimum.

Algorithm 10: Steepest Descent Minimization

```
$w^* = \textsc{SteepestDescentMinimization}(\varepsilon)$
**Input:**
    $\varepsilon > 0$ tolerance
**Output:**
    $w^* \in \mathbb{R}^D$ learned weight vector
**Algorithm:**
$w_{\mathrm{cur}} \leftarrow 0$
**repeat**
    $d \leftarrow -\nabla_w \mathcal{L}(w_{\mathrm{cur}})$        {descent direction}
    $\eta \leftarrow \operatorname{argmin}_{\eta>0} \mathcal{L}(w_{\mathrm{cur}} + \eta d)$        {univariate line search}
    $w_{\mathrm{cur}} \leftarrow w_{\mathrm{cur}} + \eta d$
**until** $\|p\| < \varepsilon$
$w^* \leftarrow w_{\mathrm{cur}}$
```

First order steepest descent is very easy to implement, and it is guaranteed to find the minimum of the objective function. However, for functions with many parameters whose contours differ strongly from a circular shape, oscillations can occur and the convergence rate is often unsatisfactory. As a consequence, many iterations are required until convergence, and overall runtime is high. Unfortunately, the situation of conditional log-likelihood optimization with small λ value is exactly the problematic case of nonquadratic objective, as can also be seen in Figure 5.1.

Second-Order Gradient Descent. Second-order gradient descent methods offer a powerful alternative to the steepest descent procedure, as they have provably better convergence rates. The most straightforward second-order gradient descent method is *Newton's method* [13]. Like the steepest descent method it performs an iterative descent, computing in each step a descent direction by multiplying the gradient vector with the inverse of the Hessian matrix. If the function to be minimize is quadratic, then the resulting descent vector points to the exact analytic minimizer. For nonquadratic functions the Newton's methods can be interpreted as forming a local quadratic approximation,



(a) $\lambda = 100$

(b) $\lambda = 10$

(c) $\lambda = 1$

(d) $\lambda = 0$

Fig. 5.1 Objective function of simplest CRF (one output node, no pairwise terms) for a training set $\{(-10,+1),(-4,+1),(6,-1),(5,-1)\}$ with $\varphi(x,+1) = (x,0)$ and $\varphi(x,-1) = (0,x)$. For large λ, contours are nearly circular and centered at the origin. With λ decreasing the regularizer loses influence and the data dependent terms start to dominate the objective function.

which is a better model that the linear approximation used in first order methods.

Second-order algorithms typically requires fewer iterations to converge compared the first order method, especially. However, this advantage comes at the expense of increased computational cost for each iteration: computing and inverting the Hessian matrix are expensive operations, and the increase in runtime of each iteration often more than outweighs the benefit one has from the reduction in the number of iterations.

To overcome the drawbacks of Newton's method, hybrid techniques have been developed that try to find similarly good descent directions without the need for storing and inverting the Hessian matrix. The currently most successful second-order methods are of the *quasi-Newton* type. Instead of computing and inverting the full Hessian in every step, they estimate the inverse matrix H_w^{-1} itself, thereby saving the expensive step of inverting a $d \times d$ matrix. This estimation is possible by iterative updates during the course of the optimization (line 15) make use only of information available from the gradient vector, such that the step of computing H_w is avoided as well. Algorithm 11 contains pseudo-code for the frequently used *Broyden–Fletcher–Goldfarb–Shanno* (BFGS) [41] procedure.

BFGS shows the same super-linear convergence speed as the Newton method does, but it requires much fewer operations in each iteration. It does, however, not overcome the problem of having to store a $d \times d$ matrix. More advanced algorithms have been developed for

Algorithm 11: Broyden–Fletcher–Goldfarb–Shanno Procedure

```
 1: $w^* = \mathrm{BFGS}(\varepsilon)$
 2: **Input:**
 3:     $\varepsilon > 0$ tolerance
 4: **Output:**
 5:     $w^* \in \mathbb{R}^D$ learned weight vector
 6: **Algorithm:**
 7: $w_{\mathrm{new}} \leftarrow 0$
 8: $B_{\mathrm{new}} \leftarrow I_{Kd}$
 9: **repeat**
10:     $(w_{\mathrm{old}}, B_{\mathrm{old}}) \leftarrow (w_{\mathrm{new}}, B_{\mathrm{new}})$
11:     $d \leftarrow B_{\mathrm{old}}^{-1} \nabla_w \mathcal{L}(w_{\mathrm{old}})$
12:     $\eta \leftarrow \mathrm{argmin}_{\eta \in \mathbb{R}} \mathcal{L}(w_{\mathrm{old}} + \eta d)$
13:     $w_{\mathrm{new}} \leftarrow w_{\mathrm{old}} + \eta d$
14:     $y \leftarrow \nabla_w \mathcal{L}(w_{\mathrm{new}}) - \nabla_w \mathcal{L}(w_{\mathrm{old}})$
15:     $B_{\mathrm{new}} \leftarrow \big(I - \frac{dy^t}{y^t d}\big) B_{\mathrm{old}} \big(I - \frac{dy^t}{y^t d}\big) + \eta \frac{dd^t}{y^t d}$
16: **until** $\|d\| < \varepsilon$
17: $w^* \leftarrow w_{\mathrm{new}}$
```

this purpose, e.g., the L-BFGS method (limited memory BFGS) [96] that stores a sequence of vector operations instead of the matrix B_k. *Conjugate gradient* [60] optimization has also successfully been applied to conditional log-likelihood minimization. A detailed discussion of these general purpose optimization techniques is beyond the scope of this monograph. Instead we refer the interested reader to the many textbooks on nonlinear optimization, e.g., [112].

5.4 Faster Training by Use of the Output Structure

Conditional random fields can be seen as a form a logistic regression classifiers (see [111]), as both share the objective of maximizing the conditional log-likelihood of the observed training pairs. However, a fundamental difference between CRFs and the ordinary multi-class logistic regression is the large size of the output space $\mathcal{Y}$. For CRFs it is typically exponentially sized in the number of output nodes of the graphical model, and it does not allow explicit enumeration. Because the closed form expression (5.14) includes a summation over *all* $y \in \mathcal{Y}$, a straightforward computation of the conditional log-likelihood or its gradient is typically computationally infeasible. In order to train a CRF using gradient descent techniques, we need to overcome this problem, typically by making use of structure in the output space, as it given, e.g., through the graphical model framework.

For tree-structured models, the sum–product algorithm (Algorithm 2) provides can be used to efficiently compute the expected value of a function over all output labels, including vector-valued functions such as $\varphi(x,y)$. Computing the gradient $\nabla_w \mathcal{L}$ is therefore computationally as costly as performing probabilistic inference. For graphical model that are not tree structured, we can run *loopy belief propagation* to approximate the expectation in Equation (5.14) and thereby obtain an approximation to the gradient. Using this approximation in a gradient descent algorithm does not guarantee convergence to the global minimum of the negative conditional log-likelihood. Nevertheless, it is a commonly used training technique for CRFs, and often yields good practical results.

5.5 Faster Training by Stochastic Example Selection

Stochastic Gradient Descent. Computing the gradient of $\mathcal{L}$ by means of Equation (5.14) require summation over all training instances, and each instance we have to perform a costly inference step. When a lot of training data is available this dependence on the number of training instances has a significant impact on the overall training time. A naive solution would be to subsample the training set before starting the optimization, but this would ignore the information contained in the discarded sample, likely reducing the prediction accuracy. Instead, one can introduce the idea of subsampling the data only into the step of computing the gradient $\nabla_w \mathcal{L}$, and compute a descent directions using formula (5.14) with only a small batch of samples, typically not more than 10. In the extremal case one can also estimate the gradient from just a single, randomly selected, training example (x^n, y^n) which yield the *stochastic gradient* approximation

$$\widetilde{\nabla}_w^{(x^n,y^n)} \mathcal{L}(w) = 2\lambda w + \varphi(x^n, y^n) - \mathbb{E}_{y \sim p(y|x^n)} \big[\varphi(x^n, y)\big]. \tag{5.16}$$

Because we want to avoid summing over all training examples, we also cannot evaluate $\mathcal{L}(w)$ and thus we cannot do a line search to determine the optimal gradient step length. Also, second-order methods like BFGS do not work well when the gradient direction is only approximate, such that we have to rely on simple first-order descent, typically with a fixed sequence of learning rates, $\eta_1, \ldots, \eta_T$ that decays over time, for example $\eta_t = \frac{\eta}{t}$ with constant $\eta > 0$. Algorithm 12 gives pseudo-code for this *Stochastic Gradient Descent (SGD)* algorithm.

SGD typically require many more iteration to converge than optimization techniques that make use of the exact gradient. However, because each iteration is several orders of magnitudes faster, stochastic training is often able to solve the training problem faster in terms of absolute runtime. The success of SGD has let to the development of many extended and improved algorithms, e.g., to avoid the need for an *a priori* choice of the parameters η by automatic gain adaption [159], and the integration of second-order information [24].

Algorithm 12: Stochastic Gradient Descent

```
$w^* = \textsc{StochasticGradientDescent}(T, \eta)$
**Input:**
    $T$ number of iterations
    $\eta_1, \dots, \eta_T$ sequence of learning rates
**Output:**
    $w^* \in \mathbb{R}^D$ learned weight vector
**Algorithm:**
$w_{\text{cur}} \leftarrow 0$
**for** t=1,...,T **do**
    $(x^n, y^n) \leftarrow$ randomly chosen training pair
    $d \leftarrow -\widetilde{\nabla}_w^{(x^n,y^n)}\mathcal{L}(w_{\text{cur}})$
    $w_{\text{cur}} \leftarrow w_{\text{cur}} + \eta_t d$
**end for**
$w^* \leftarrow w_{\text{cur}}$
```

5.6 Faster Training by Stochastic Gradient Approximation

When other approaches fail, *sampling* methods often still offer a viable alternative, as they provide a universal tool for evaluating expectations over random variables. From Equation (5.14) we know that the computationally hard part in computing the gradient $\nabla_w\mathcal{L}$ has the form of the expectation of $\varphi(x,y)$ with respect to the distribution $p(y|x,w)$. If we have a method to obtain $i.i.d.$ samples $\mathcal{S} = \{y^{(1)}, \dots, y^{(S)}\}$ from this distribution, we can form an unbiased estimator of $\mathbb{E}_{y\sim p(y|x^n)}[\varphi(x^n,y)]$ by $\frac{1}{S}\sum_{i=1}^{S}\varphi(x,y^{(i)})$. Inserting this into Equation (5.16) we obtain an unbiased estimator of $\nabla_w\mathcal{L}(x,w)$, where the law of large numbers guarantees convergence of the approximation to the exact gradient with a rate of $\frac{1}{\sqrt{S}}$. Consequently, any procedure to sample from $p(y|x^n)$ for $n = 1, \dots, N$ provides us with a tool for estimating $\nabla_w\mathcal{L}(x,w)$. In particular, we can use *Markov chain Monte Carlo (MCMC)* sampling that we had introduced in Section 3.4.1. However, the computational cost of this setup is still very high, since we need a full MCMC run for each training example for each gradient computation. One also has to consider that even after the MCMC sampler has converged to its equilibrium distribution, we obtain only approximations of the gradient $\nabla_w \mathcal{L}$. This is an inherent limitation of all sampling-based approaches, not a problem of a specific MCMC sampler.

As we had seen before, inexact gradient directions pose problems for most second-order gradient descent methods, such that one has to rely on first-order gradient descent to optimize the conditional log-likelihood with sampling based methods. Consequently, one will likely need many iterations, which justified additional effort to accelerate each iteration as much as possible. This is possible because of a crucial insight by Hinton [61]: instead of waiting for the Markov chain to converge against its equilibrium distribution before computing a gradient approximation, we can use samples from the very beginning of the Markov chain. This will result in a worse approximate gradient, but strongly reduced runtime to compute it.

An algorithm that takes this observation to the extreme is *contrastive divergence* training as stated in Algorithm 13. It performs Gibbs sampling without burn-in, estimating the gradient from only a single sample. This provides a very noise estimate of the best descent direction, and consequently very many iterations might be required until convergence. However, this drawback is more than compensated by the fact that the computation time of each iteration is reduced. Contrastive divergence training has proved a competitive training technique for large scale CRFs [58]. Despite the many simplifications compared to a complete sampling approach, there are theoretic results that guarantee convergence to the global minimum under certain conditions [173], and that characterize the role of the sampling scheme used in the contrastive divergence procedure [7].

5.7 Faster Training by Two-Stage Training

The feature maps $\varphi_f(y_f, x)$ used in CRFs for computer vision tasks play different role depending on whether the factors are unary, pairwise or of higher order. Unary factors have typically high-dimensional features maps. In combination with their weight vectors, they give strong cues about what labels are likely for each individual node. Pairwise and higher order factors have typically low-dimensional or even scalar

Algorithm 13: Contrastive Divergence (CD) Training

```
$w^* = \textsc{ContrastiveDivergence}(T, \eta_1, \ldots, \eta_T)$
Input:
    $T$ number of iterations
    $\eta_1, \ldots, \eta_T$ sequence of learning rates
Output:
    $w^* \in \mathbb{R}^D$ learned weight vector
Algorithm:
$w_{\text{cur}} \leftarrow 0$
for t=1,...,T do
    $(x^n, y^n) \leftarrow$ randomly chosen training pair
    $\hat{y} \leftarrow$ Gibbs sample from $p(y|x^n, w_{\text{cur}})$ initialized at $y^n$
    $d \leftarrow \lambda w_{\text{cur}} + \varphi(x^n, y^n) - \varphi(x^n, \hat{y})$.
    $w_{\text{cur}} \leftarrow w_{\text{cur}} + \eta_t d$
end for
$w^* \leftarrow w_{\text{cur}}$
```

feature maps, such as the Potts feature $\varphi(y_i, y_j) = [y_i \neq y_j]$ that is commonly used to encode label smoothness for image segmentation. In combination, CRF training is a high-dimensional optimization problem, but most of the free parameters encode only per-node information, which might as well be learned by a simpler per-node training technique.

We can utilize the above observation by training CRFs in computer vision in a two-stage procedure: first, we train independent per-node classifiers f^{y_i}, e.g., by logistic regression or a support vector machine. The output of these classifiers we use as one-dimensional feature map for unary factors, $\varphi_i(y_i, x) = f_i^{y_i}(x)$. Pairwise and higher order feature maps are chosen as for conventional CRF training, and a standard CRF training technique is applied. The main advantage of such two-stage training is a significant reduction in training time, because the number of free parameters in the structured training step is reduced. A further advantage is a gain in flexibility in the first stage: we can use arbitrary classifiers to learn the per-node features f^{y_i}, whereas a probabilistically trained CRF selects the weights of the unary features always as the result of probabilistically training a log-linear model, which resembles a logistic regression classifier.

The disadvantage of pretraining is a loss of expressive power for the second stage: only the magnitude of importance of the unary features can be adjusted, while the high-dimensional weight vector remains fixed.

Example 5.1 (Figure-Ground Image Segmentation). Fulkerson et al. [47] introduce a conditional random field on the superpixels level for the classical problem of figure-ground segmentation, i.e., distinguishing which pixels in an image belong to an object and which belong to the background. To reduce the number of model parameter they set up the model for two-stage learning: for the unary potentials, they first train a support vector machine with χ^2-RBF kernel using a bag of visual word representation of the superpixels and their neighbors as inputs. The SVM outputs are normalized into the range $[0,1]$ by Platt-scaling [119] and become one-dimensional features. The pairwise terms are Potts potentials weighted by the image gradient and therefore also one-dimensional. By giving all nodes a constant weight and letting all edges of the graphical model share weights, the CRF can be written with only a single free parameter: a tradeoff between unary and pairwise terms. This value is learned by cross-validation on the training set.

5.8 Latent Variables

Some prediction tasks are easier to model by introducing *latent variables*, i.e., quantities that are observed neither at training nor at test time. For example, part-based object detection makes us of the fact that the decision where an object, such as a car, is located is much easier once we know the location of its characteristic parts, e.g., its wheels. Part locations are typically not specified in the training data and therefore is therefore modeled most adequately by latent variables.

In a probabilistic model, latent variables do not pose a principled problem, one simply treats them as additional output variables $z \in \mathcal{Z}$ which are not specified in the training set. The CRF models the joint conditional probability distribution of observable and unobserved outputs $p(y,z|x,w)$. Regularized conditional likelihood maximization for a training set, $\mathcal{D} = \{(x^n,y^n)\}_{n=1,\dots,N}$ still consists of minimizing

$$\mathcal{L}(w) = \lambda\|w\|^2 - \sum_{n=1}^{N} \log p(y^n|x^n,w). \tag{5.17}$$

Using the marginalization rule $p(y|x,w) = \sum_z p(y,z|x,w)$ we can express this in terms of the CRF distribution

$$\begin{aligned} &= \lambda\|w\|^2 - \sum_{n=1}^{N} \log \sum_{z\in\mathcal{Z}} \exp(-\langle w, \varphi(x^n,y^n,z)\rangle) \\ &\quad + \sum_{n=1}^{N} \log \sum_{\substack{z\in\mathcal{Z}\\ y\in\mathcal{Y}}} \exp(-\langle w, \varphi(x^n,y,z)\rangle), \end{aligned} \tag{5.18}$$

and we can compute the gradient

$$\begin{aligned} \nabla_w \mathcal{L}(w) = 2\lambda w &+ \sum_{n=1}^{N} \mathbb{E}_{z\sim p(z|x,y,w)}[\varphi(x^n,y^n,z)] \\ &- \sum_{n=1}^{N} \mathbb{E}_{(y,z)\sim p(y,z|x,w)}[\varphi(x^n,y,z)]. \end{aligned} \tag{5.19}$$

In contrast to the fully observed situation, $p(y|x,w)$ is not log-linear, but a mixture of many log-linear models. This has the effect that solving Equation (5.17) is not convex, and it is no longer guaranteed that gradient-descent optimization will find the global optimum.

Expectation maximization. In searching alternative optimization strategies it has been observed that the classical *expectation maximization (EM)* algorithm can be applied to latent variable CRFs if certain independence conditions between observed and latent variables hold (see [77, Section 19] for details). EM-based training iterates two steps: for a current parameter $\hat{w}$ it forms expressions $q^n(z)$ that is the distribution $p(z|x^n,y^n,\hat{w})$ seen as only a function of z (E-step). Subsequently, it computes a new estimate $\hat{w}$ by solving the auxiliary problem

$\hat{w} = \operatorname{argmin}_{w \in \mathbb{R}^D} \mathcal{L}_{\mathrm{EM}}(w)$ with

$$\mathcal{L}_{\mathrm{EM}}(w) = \lambda \|w\|^2 - \sum_{n=1}^{N} \sum_{z \in \mathcal{Z}} q^n(z) \log p(y^n, z | x^n, w). \tag{5.20}$$

(M-step). Although formulated in a different way, direct gradient descent on Equation (5.17) and EM-based training are very related, as can be seen by computing the gradient of Equation (5.20)

$$\begin{aligned}\nabla_w \mathcal{L}_{EM}(w) = 2\lambda w &+ \sum_{n=1}^{N} \mathbb{E}_{z \sim p(z|x,y,\hat{w})} [\varphi(x^n, y^n, z)] \\ &- \sum_{n=1}^{N} \mathbb{E}_{z \sim p(z|x,y,\hat{w})} \mathbb{E}_{y \sim p(y|z,x,w)} [\varphi(x^n, y, z)].\end{aligned} \tag{5.21}$$

It differs from the log-likelihood gradient (5.19) only in the fact that the expectations of the latent variables z are taken with respect to a distribution parameterized by the weight vector $\hat{w}$ of the previous iteration.

An interesting aspect of both, gradient-based and EM-based training, is that they can be expressed using only expectation operations over the feature functions. This allows easy integration of sampling based approximations, such as contrastive divergence. Nevertheless, latent variable CRFs have found relatively little application in computer vision tasks so far, possibly because of the increased computational cost of marginalizing out potentially many latent variables, and because the nonconvexity of the problem requires a good initialization to avoid convergence to a suboptimal local minimum.

Example 5.2 (Gesture Recognition). Morency et al. [106] perform gesture recognition using a conditional random with chain-structured latent variables, see Figure 5.2. The inputs consist of a sequence of images, and each of these also has an output variable that indicates the presence of a certain gesture, e.g., *nod* or *smile.* Input and output nodes are connected through a layer of latent nodes. Because these have a larger state-space than the output nodes, they can reflect *sub-stages*

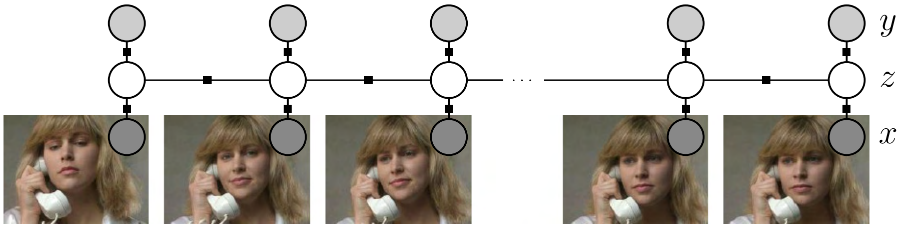


Fig. 5.2 Gesture recognition with a chain-structured latent variable conditional random field [106].(Image Source: Stanford University).

of a gesture, e.g., the *down* and *up* part of a nod. The model is trained using gradient based BFGS optimization, which is possible efficiently because the model is loop-free.

5.9 Other Training Objectives

As we have seen in the previous sections, maximizing the conditional likelihood for loopy graphical models is a computationally expensive task. In particular, computing the exact gradient of the conditional log-likelihood is often infeasible, and even finding good approximations can be computationally expensive, even when making use of all available acceleration possibilities. Since the time available for training is limited, it is common stop gradient based conditional likelihood maximization before convergence, and one settles for a weight vector that only approximately minimizes the posterior distribution $p(w|\mathcal{D})$.

It stands to reason to look for other training objectives that also approximate to the conditional likelihood, but could be more efficient to compute. In particular we can approximate $p(y|x,w)$ itself with a simpler distribution $p_{\text{approx}}(y|x,w)$, for which the gradient can be computed more efficiently and therefore the global optimum of $p_{\text{approx}}(w|\mathcal{D})$ can be found. A popular technique of this kind is PL [16].

Definition 5.2 (Pseudo-Likelihood Training). Let $p(y|x,w) = \frac{1}{Z(x,w)} \exp\langle w, \varphi(x,y)\rangle$ be a probability distribution parameterized by $w \in \mathbb{R}^D$, and let $\mathcal{D} = \{(x^n, y^n)\}_{n=1,\ldots,N}$ be a set of training examples.

PL training chooses the parameter as

$$w^*_{\text{PL}} = \operatorname*{argmin}_{w \in \mathbb{R}^D} \ \lambda \|w\|^2 + \sum_{n=1}^{N} \sum_{s=1}^{M} [\langle w, \varphi(x^n, y^n) \rangle + \log Z_s(x^n, y^n_{\neg s}, w)] \tag{5.22}$$

with $y^n_{\neg s} = (y^n_1, \ldots, y^n_{s-1}, y^n_{s+1}, \ldots, y^n_M)$ and

$$Z_s(x, y^n_{\neg s}, w) = \sum_{y_s \in \mathcal{Y}_s} \exp\big(- \langle w, \varphi(x, y^n_{\neg s}, y_s) \rangle \big) 0 \tag{5.23}$$

PL training is inspired by the observation that maximizing $p(y|x)$ can easily be solved if every output site y_s depended only on observed quantities and not on any other unknowns, because under this assumption the optimal value of each output node can be found by an independent maximization. During training, one can simulate this situation, because x and y are both observed and define the PL of an input–output pair (x, y) as the product of the conditional likelihoods of all individual output nodes conditioned on the remaining nodes:

$$p_{\text{PL}}(y|x, w) := \prod_{s=1}^{M} p_{\text{PL}}(y_s | y_{\neg s}, x, w), \tag{5.24}$$

where $p_{\text{PL}}(y_s|y_{\neg s}, x, w)$ is given by the loglinear CRF model

$$p_{\text{PL}}(y_s|y_{\neg s}, x, w) = \frac{1}{Z_s(x, y_{\neg s}, w)} \exp(\langle w, \varphi(x, y) \rangle), \tag{5.25}$$

where $Z_s(x, y_{\neg s}, w)$ has the form of Equation (5.23).

Rule (5.22) follows from this by demanding that w^*_{PL} should maximize the regularized PL of the training set $\mathcal{D}$. Denoting by $\mathcal{L}_{\text{PL}}(w)$ the objective function of Equation (5.22) and calculating its gradient, we obtain

$$\nabla_w \mathcal{L}_{\text{PL}}(w) = 2\lambda w + \sum_{n=1}^{N} \sum_{s=1}^{M} \varphi(x^n, y^n) - \mathbb{E}_{y_s \sim p_{\text{PL}}(y_s|y_{\neg s}, x, w)} \big[\varphi(x^n, y^n_{\neg s}, y_s) \big], \tag{5.26}$$

which is much easier to compute than the full log-likelihood gradient (5.14), because the expectations in the last term are only over a single random variable at a time. Consequently, PL is nearly as efficient as training a model with independent outputs nodes.

For prediction, we cannot rely on the same trick of conditioning on parts of the output, because no component of y is observed at that time. Instead, one uses the learned weight vector w^*_{PL} in combination with the previously introduced (approximate) inference techniques, e.g., (loopy) belief propagation.

Despite this mismatch between training objective and test setup, PL training has been shown to be consistent under certain conditions, i.e., given infinite amounts of training data, it leads to the same weight vector as exact conditional likelihood maximization [16]. Even for the realistic case of finite data, PL training has been used successfully for different computer vision tasks. However, it has been observed that PL training tends to lead to low generalization performance when there are strong dependencies between the output sites.

5.9.1 Piecewise Training

Piecewise (PW) training [144] has been developed as an alternative way to make the maximization of the conditional likelihood manageable while avoiding the statistical inefficiency of PL training.

Definition 5.3 (Piecewise Training). Let $p(y|x,w) = \frac{1}{Z(x,w)} \exp\langle w, \varphi(x,y)\rangle$ be a probability distribution parameterized by $w \in \mathbb{R}^D$, and let $\mathcal{D} = \{(x^n,y^n)\}_{n=1,\dots,N}$ be a set of training examples. Let $\mathcal{F}$ be the set of factors in a graphical model representation of p, such that $\varphi(x,y) = (\varphi_F(x_F,y_F))_{F\in\mathcal{F}}$. *Piecewise* training chooses the parameters as $w^*_{\text{PW}} = (w^*_F)_{F\in\mathcal{F}}$ with

$$w^*_F = \operatorname*{argmin}_{w\in\mathbb{R}^D} \; \lambda\|w\|^2 + \sum_{n=1}^{N} \langle w_F, \varphi_F(x^n_F, y^n_F)\rangle + \log Z_F(x^n_F, y^n_F, w_F), \tag{5.27}$$

with

$$Z_F(x) = \sum_{y_F \in Y_F} \exp(-\langle w_F, \varphi_F(x_F, y_F)\rangle). \tag{5.28}$$

The piecewise training rule is justified by approximating $p(y|x,w)$ by a distribution that is a product over the factors

$$p_{\text{PW}}(y|x,w) := \prod_{F \in \mathcal{F}} p(y_F|x_F, w_F), \tag{5.29}$$

where

$$p(y_F|x) = \frac{1}{Z_F(x, w_F)} \exp(-\langle w_F, \varphi_F(x_F, y_F)\rangle). \tag{5.30}$$

If we maximize this expression with regularization over the training set $\mathcal{D}$, one observes that the optimization problem decouples over the factors and one obtains Equation (5.27).

The piecewise approximation does not suffer from the same problems as PL, because it does not condition on the output variables during training. At the same time, it retains more expressive power than a model with completely independent label sites, because it can model the dependence between output sites by making use of factors that contain more than one output variable. Comparing $p_{\text{PW}}(y|x,w)$ with the exact expression for $p(y|x,w)$, we see that both models differ only in their expression for the partition function. While the exact $Z(w)$ does not factorize into a product of simpler terms, its piecewise approximation $Z_{\text{PW}}(w)$ factorizes over the set of factors.

Consequently, we can perform gradient based training of Equation (5.27) for each factor as long as the individual factors remain small, i.e., contain not more than 2 to 5 output sites. Note that the possibility of independent training holds in particular also for the unary factors. This means that the simplification made by piece-wise training of CRFs resemble two-stage training of Section 5.7.

Similar to PL training, the situation at evaluation time differs from the training setup, because we cannot enforce a factorization there. Instead, we concatenate the learned weight vectors w_F^* into a joint vector w^*_{PW} and apply a standard inference technique to the ordinary conditional distribution $p(y|x, w^*_{\text{PW}})$. However, practical experiments suggest that, in computer vision tasks, two-stage training does not incur a significant loss in prediction accuracy [113].

6

Structured Support Vector Machines

Besides probabilistically trained CRF, *margin-based parameter learning* has recently gained a lot of popularity in computer vision. It aims at solving *Problem 4* of *Graphical Models*. Throughout this section we assume that the learning task we try to solve has a fixed (but unknown) probability density $d(x,y)$ from we have an *i.i.d.* sample set $\mathcal{D} = \{(x^1,y^1),\dots,(x^n,y^n)\} \subset \mathcal{X} \times \mathcal{Y}$ that we will use for training. Furthermore, we assume that we fixed a loss function $\Delta\colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$, where $\Delta(y,y')$ specifies the cost of predicting y' for a sample when the correct label is y. For convenience in notation, we assume that a correct prediction incurs no loss, i.e., $\Delta(y,y) = 0$.

6.1 Structural Risk Minimization

In trying to solve Problem 4, we aim at finding a prediction function $f\colon \mathcal{X} \to \mathcal{Y}$ which minimizes the *Bayes risk*, i.e., the expected Δ-loss $\mathbb{E}_{(x,y)\sim d(x,y)}\Delta(y,f(x))$. As in *Structured Prediction* we will assume that f has the form $f(x) = \operatorname{argmax}_y g(x,y,w)$ for an auxiliary evaluation function $g\colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, which is parameterized by $w \in \mathbb{R}^D$. This is an increase in flexibility over CRF, where we considered only the case of $g(x,y,w) = \log p(y|x,w)$ for a conditional probability distribution p approximating d.

Because $d(x,y)$ is unknown, minimizing the Bayes risk directly is not possible, but *structural risk minimization* [154] offers an indirect way to identify a function with good predictive qualities. It chooses a prediction function f that minimizes the *regularized empirical risk* functional

$$\mathcal{R}(f) + \frac{C}{N}\sum_{n=1}^{N}\Delta(y^n, f(x^n)), \tag{6.1}$$

where the second term is an empirical estimate of the expected risk, and the first term is chosen as a *regularizer* that prevents overfitting by penalizing functions depending on how complex they are. For structured prediction functions of the form $f(x) = \operatorname{argmax}_y g(x,y,w)$ minimizing the expression (6.1) numerically with respect to w is typically infeasible, because the term $\Delta(y^n, f(x))$ is piece-wise constant and this renders gradient-based techniques useless. However, results from statistical learning theory show that it can be sufficient to minimize a *convex upper bound* to (6.1) and still achieve an optimal prediction accuracy in the limit of infinite data [176].[1] This forms the basis of *structured support vector machine (S-SVM)* training.

Definition 6.1 (Structured Support Vector Machine). Let $g(x,y,w) = \langle w, \varphi(x,y)\rangle$ be a compatibility function parameterized by $w \in \mathbb{R}^D$. For any $C > 0$, *structured support vector machine (S-SVM)* training chooses the parameter

$$w^* = \operatorname*{argmin}_{w\in\mathbb{R}^D} \; \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} \; \ell(x^n, y^n, w), \tag{6.2}$$

with

$$\ell(x^n, y^n, w) = \max_{y\in\mathcal{Y}} \; \Delta(y^n, y) - g(x^n, y^n, w) + g(x^n, y, w). \tag{6.3}$$

[1] Unfortunately, the most satisfactory *consistency* property does not hold for the general multi-class or structured prediction situations [91]. Optimizing a convex upper bound nevertheless makes sense, see [101] for an in-depth discussion.

Equation (6.2) is derived from Equation (6.1) by choosing $\mathcal{R}(f) = \frac{1}{2}\|w\|^2$ as a regularizer and replacing the Δ-loss by its convex upper bound ℓ. To show the bounding property we observe that for any $f(x) = \operatorname{argmax}_y g(x,y,w)$ we have

$$\Delta(y^n, f(x^n)) \leq \Delta(y^n, f(x^n)) - g(x^n, y^n, w) + g(x^n, f(x^n), w) \qquad (6.4)$$

$$\leq \max_y \; \Delta(y^n, y) - g(x^n, y^n, w) + g(x^n, y, w) \qquad (6.5)$$

$$= \ell(x^n, y^n, w). \qquad (6.6)$$

The convexity follows because ℓ is the maximum over many functions that are affine in w.

Equation (6.3) generalizes the *Hinge loss* to multiple outputs labels [148].[2] As a result, Equation (6.2) can be interpreted as a *maximum margin* training procedure that extends the popular *support vector machine* classifiers to structured output spaces. The name *structured support vector machine* learning [151] stems from this observation.

As for conditional random fields, training structured output support vector machines is a computationally expensive task, and it often only becomes feasible by a careful analysis of the problem components and the exploitation of domain knowledge. Before we study these aspects of S-SVM training in more detail we introduce some structured loss functions that are commonly used in computer vision problems.

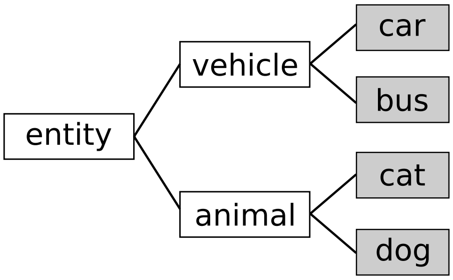
Example 6.1 (Structured Loss Functions). Different structured prediction tasks come with different loss functions to judge whether the prediction made for a training input is *good*, or *similar enough* to the training output. See Figures 6.1 and 6.2 for illustrations.

Zero-one loss: $\Delta(y, y') = [\![y \neq y']\!]$. This is the most common loss for multi-class problems but less frequently used for structured prediction

[2] This extension is generally called the *margin-rescaled* Hinge loss. As an alternative upper bound the *slack-rescaled* Hinge loss

$$\ell(x^n, y^n, w) = \left[\max_{y \in \mathcal{Y}} \; \Delta(y^n, y)\big(1 - g(x^n, y^n, w) + g(x^n, y, w)\big)\right]_+ .$$

has been proposed by Tsochantaridis et al. [151], but this has found less use in computer vision as it leads to a more complicated optimization problem, see [128].



Zero-One loss:
$\Delta_{0/1}(\mathtt{cat}, \mathtt{cat}) = 0$
$\Delta_{0/1}(\mathtt{cat}, \mathtt{dog}) = 1$
$\Delta_{0/1}(\mathtt{cat}, \mathtt{bus}) = 1$

Hierarchical loss:
$\Delta_H(\mathtt{cat}, \mathtt{cat}) = 0$
$\Delta_H(\mathtt{cat}, \mathtt{dog}) = 1$
$\Delta_H(\mathtt{cat}, \mathtt{bus}) = 2$

Fig. 6.1 Multi-class loss functions. Zero-one loss $\Delta_{0/1}$ penalizes all mistakes equally. Hierarchical loss Δ_H penalizes mistakes depending on the tree distance between classes in a hierarchy.

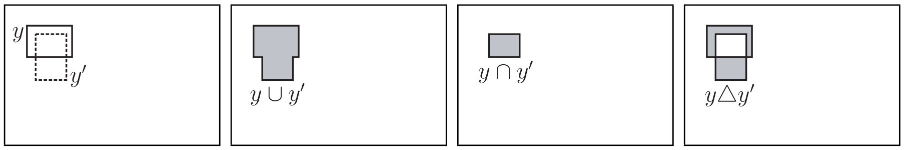


Fig. 6.2 Region-based loss functions. Left: target region y (solid boundary) and prediction y' (dashed boundary). Middle: union $y \cup y'$ and intersection $y \cap y'$ of the regions. Measured by area overlap loss, y' is a rather poor prediction of y, because taking the ratio of areas results in $\Delta(y, y') \approx \frac{2}{3}$. Right: set of incorrectly predicted pixel $y \triangle y'$. Measured by per-pixel Hamming loss, y' is a good prediction of y ($\Delta(y, y') \approx 0.03$), because most image pixels are correctly classified as belonging to the background.

tasks with large output spaces. Every prediction that is not fully identical to the intended one is considered a mistakes, and all mistakes are penalized equally.

Hierarchical multi-class loss: $\Delta(y, y') = \frac{1}{2}\mathrm{dist}_H(y, y')$, where H is a hierarchy over the classes in $\mathcal{Y}$ and $\mathrm{dist}_H(y, y')$ measures the distance between y and y' within the hierarchy. This loss is a common way to incorporate information about label hierarchies in multi-class prediction problems. Differences between predicted and correct label are penalized less if they occur between similar classes (with respect to the given hierarchy) than if the occur between dissimilar ones. This is shown in Figure 6.1.

Hamming loss: $\Delta(y, y') = \frac{1}{m}\sum_{i=1}^{m} [\![y_i \neq y'_i]\!]$. Frequently used loss for image segmentation or other tasks in which the output y consists of multiple part labels $y_1, \dots, y_m$. Each part label is judged independently and the average number of labeling errors is determined.

Area overlap: $\Delta(y, y') = \frac{\text{area}(y \cap y')}{\text{area}(y \cup y')}$. Standard loss in *object localization*, e.g., in the PASCAL detection challenges, with y and y' being bounding box coordinates, and $y \cap y'$ and $y \cup y'$ are their intersection and union, respectively. This is shown in Figure 6.2.

6.2 Numeric Optimization

All summands in Equation (6.2) are convex functions in w, and therefore S-SVM training is a convex optimization problem. This hold even for the limit case of $C \to \infty$, which is only solvable with finite objective value if a weight vector with zero loss exists. As in the CRF situation we can apply standard convex optimization methods to find its global minimum. Figure 6.3 shows an illustration of the objective function in a simplified situation and the effect of different C values. Note that the contours are similar to the CRF situation[3] (Figure 5.1 on page 315) except that they are not everywhere differentiable, which is the result of the max operation in Equation (6.3).

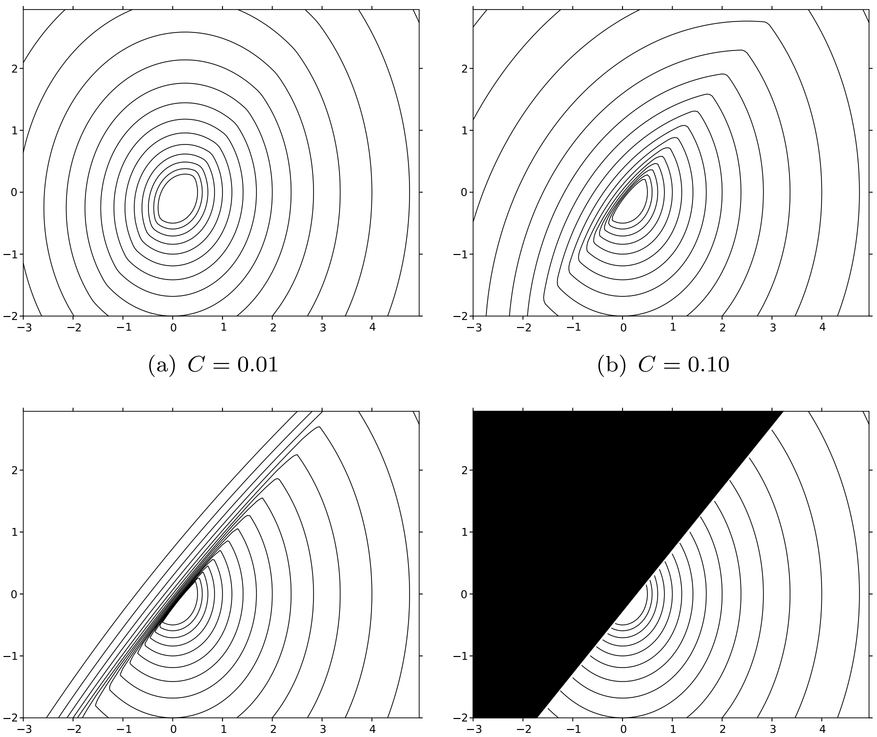
Subgradient Descent Minimization. Because of its non-differentiability, Equation (6.2) cannot be solved by straight-forward gradient descent optimization. However, *subgradient methods* [139] that we introduced in Section 4.7.3 are suitable candidates.

Algorithm 14 contains pseudo-code for S-SVM training by subgradient descent. For each summand of the loss term, line 11 identifies which $y \in \mathcal{Y}$ is active in the max operation of Equation (6.3). We can ignore the summand $\langle w, \varphi(x^n, y^n) \rangle$ of $\ell(x^n, y^n, w_{\text{cur}})$ for this because it does

[3] This similarity is of course not just a coincidence. In fact, if we set $\Delta(y, y') = [y \neq y']$ and use the *multiclass logistic loss*,

$$\ell(x^n, y^n, w) = \log \sum_y \exp(g(x^n, y^n, w) - g(x^n, y, w)).$$

instead of the Hinge loss in Equation (6.2), we recover an optimization problem equivalent to *conditional random field training* (5.2) with $\lambda = \frac{N}{2C}$. This illustrates that for good ideas there is often more than one way to justify them. For further connections between CRFs and S-SVMs, see e.g., [120].



(a) $C = 0.01$

(b) $C = 0.10$

(c) $C = 1.00$

(d) $C \to \infty$

Fig. 6.3 S-SVM objective function for $\mathcal{X} = \mathbb{R}$ and $\mathcal{Y} = \{-1,+1\}$ with training set $\{(-10,+1),(-4,+1),(6,-1),(5,-1)\}$ and $\varphi(x,+1) = (x,0)$ and $\varphi(x,-1) = (0,x)$. For small C, contours are nearly circular and centered the origin. With C increasing, the loss term gains more influence and the regions of non-differentiability become more visible. For $C \to \infty$ a large part of the parameter space causes infinite objective value (black region).

not depend on y. Line 13 computes the subgradient of $\ell(x^n, y^n, w_{\text{cur}})$ with respect to w, and line 14 updates the weight vector according to the chosen learning rate.

In order to perform one weight vector update, we have to solve n optimization problems of the form $\operatorname{argmax}_y \Delta(y^n, y) + g(x^n, y, w)$. We call these *loss-augmented prediction* steps because of their strong resemblance to the evaluation of $f(x) = \operatorname{argmax}_y g(x,y,w)$. In fact, in many cases Δ can be rewritten to look like additional terms of the inner product evaluation required to compute F. In this case we can reuse the routines for MAP prediction of *Structured Prediction* to perform the

Algorithm 14: Subgradient Descent SSVM Training

```
$w^* = \textsc{SubgradientDescent}(T, \eta)$
**Input:**
    $T$ number of iterations
    $\eta$ learning rate
**Output:**
    $w^* \in \mathbb{R}^D$ learned weight vector
**Algorithm:**
$w_{\text{cur}} \leftarrow 0$
**for** t=1,...,T **do**
    **for** n=1,...,N **do**
        $\hat{y}^n = \operatorname{argmax}_{y \in \mathcal{Y}} \Delta(y^n, y) + \langle w, \varphi(x^n, y) \rangle$
    **end for**
    $p \leftarrow w_{\text{cur}} + \frac{C}{N} \sum_{n=1}^{N} \left[ \varphi(x^n, y^n) - \varphi(x^n, \hat{y}^n) \right]$
    $w_{\text{cur}} \leftarrow w_{\text{cur}} - \frac{\eta}{t} p$
**end for**
$w^* \leftarrow w_{\text{cur}}$
```

maximization step and only one prediction routine will be necessary which is then used during training as well as for structured prediction. We call training algorithms with this property *prediction-based*. CRF training of CRF is not of this kind, as it requires probabilistic inference during training.

Subgradient descent optimization is easy to implement and it can be applied for batch as well as for online learning. However, it has a relatively weak convergence rate of $O(\sqrt{\varepsilon})$ [109], i.e., reducing the distance between w_{cur} and w^* by a factor of ε can require $O(\frac{1}{\varepsilon^2})$ iterations. Given that each iteration requires multiple costly steps of loss-augmented prediction, it is natural to look for efficient way to optimize Equation (6.2). A promising candidate for this is the *bundle method*, which improves the convergence rate to $O(\frac{1}{\varepsilon})$ [149]. However, there is no practical experience with this training technique in a computer vision context so far.

Example 6.2 (Learning to Plan). *Learning to plan* is the task of predicting paths through aerial images for given start and end points

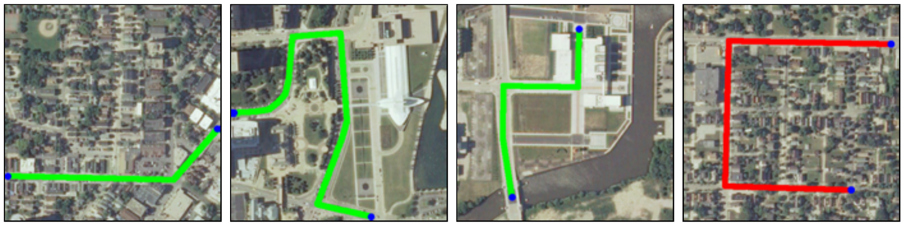


Fig. 6.4 Illustration of *Learning to Plan*. Given a set of training paths (green) between given start and end locations, the task consists of learning a prediction function that find similar paths in new images (red), e.g., preferring larger roads over smaller ones and avoiding water and vegetation. (Image Source: WisconsinView.org, SSEC).

(see Figure 6.4 for an illustration). Ratliff et al. [123] propose *maximum margin planning* for this task: they use a structured support vector machine to learn a prediction function for planning: training data consists of images in a per-pixel color representation, and example paths with the intended property, e.g., preferring roads and avoiding vegetation. The S-SVM optimization problem is solved using subgradient descent, where for prediction and loss-augmented prediction the A^* algorithm [56] is used.

Working Set Training. It is possible to express the S-SVM training problem in a way that avoids the complications introduced by the non-differentiability of the loss function.

Definition 6.2 (S-SVM — Formulation with Slack Variables). Let $\xi = (\xi^1,\dots,\xi^n) \in \mathbb{R}^N_+$ be a vector of auxiliary variables, called *slack variables*. For any $C > 0$ the *slack formulation of S-SVM training* chooses the parameter w^* by solving

$$(w^*,\xi^*) = \operatorname*{argmin}_{w\in\mathbb{R}^D,\xi\in\mathbb{R}^n_+} \quad \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} \xi^n \tag{6.7}$$

subject to, for $n = 1,\dots,N$:

$$g(x^n,y^n,w) - g(x^n,y,w) \geq \Delta(y^n,y) - \xi^n, \quad \text{for all } y \in \mathcal{Y}. \tag{6.8}$$

The slack formulation of S-SVM training is equivalent to the *loss-formulation of S-SVM training* given by Equation (6.2), i.e., both return the same weight vector w^*. To see this, one observes that the constraints (6.8) ensure $\xi^n \geq \ell(x^n, y^n, w)$ for all $n = 1, \ldots, N$, while Equation (6.7) aims at minimizing the value of ξ^n. At their optimal values, the slack variables have the same values as the corresponding loss terms, and consequently, the optimal weight vectors between both formulation coincide as well.

From the optimization point of view, the constrained optimization problem (6.7)/(6.8) has a more elementary form than Equation (6.2). Its objective function is quadratic in w and linear in ξ, and all constraints are linear, thereby making the optimization problem jointly convex. However, this advantage comes at the expense of a large number of constraints, namely $|\mathcal{Y}|$ inequalities per training sample. For most structured prediction problems, this number is far larger than what software packages for constrained convex optimization can process in reasonable time. Typically, it is not even possible to store all constraints explicitly in memory. However, because there are only D degrees of freedom in the weight vector, and N degrees of freedom in the slack variables, it is possible to show that $D + N$ constraints suffice to determine the optimal solution, whereas the others will be fulfilled automatically. If we knew the set of relevant constraints in advance we could solve the optimization (6.7)/(6.8) efficiently. This motivates the use of *cutting plane* training [67], for which pseudo-code is given in Algorithm 15.

Cutting plane training is a delayed constraint generation technique. It searches for the best weight vector and the set of active constraints simultaneously in an iterative manner. Starting from an empty working set, in each iteration it solves the optimization problem (6.7)/(6.8) with only the constraints of the working set (line 9). Subsequently, it checks for each sample if any of the $|\mathcal{Y}|$ constraints are violated (line (11)). If not, one has found the optimal solution and the algorithm terminates. Otherwise it adds the most violated constraints to the working set (line 13) and starts the next iteration. To achieve faster convergence one typically chooses a tolerance $\varepsilon > 0$ and requires a constraint to be violated by at least ε in order to be included in the working set. It is

Algorithm 15: Cutting Plane S-SVM Training

```
1:  $w^* = \textsc{CuttingPlane}(\varepsilon)$
2:  Input:
3:      $\varepsilon$ tolerance
4:  Output:
5:      $w^* \in \mathbb{R}^D$ learned weight vector
6:  Algorithm:
7:  $S \leftarrow \emptyset$
8:  repeat
9:      $(w_{\text{cur}}, \xi_{\text{cur}}) \leftarrow$ solution to (6.7) with constraints (6.8) from $S$
10:     for $n = 1, \ldots, N$ do
11:         $y^* \leftarrow \text{argmax}_{y \in \mathcal{Y}} H_n(y; w_{\text{cur}}, \xi_{\text{cur}})$
12:         if $H_n(y^*; w_{\text{cur}}, \xi_{\text{cur}}) > \varepsilon$ then
13:             $S \leftarrow S \cup \{(x^n, y^*)\}$
14:         end if
15:     end for
16: until $S$ did not change in this iteration
17: $w^* \leftarrow w_{\text{cur}}$
    where $H_n(y; w, \xi) := g(x^n, y, w) - g(x^n, y^n, w) + \Delta(y, y^n) - \xi^n$.
```

then possible to prove convergence after $O(\frac{1}{\varepsilon^2})$ steps with the guarantee that the objective value at the solution differs at most ε from the global mimimum [151].

In order to train an S-SVM using Algorithm 15 we need be able to perform two kinds of operations: solving the quadratic optimization problem in line 9, and identifying the potentially most violated constraint in line 11. As long as the working set size is reasonable, the first task can be solved using general purpose quadratic program solvers, either directly or after dualizing it. It is, however, also possible to adapt existing SVM training methods and this typically leads to much higher performance [118]. The second task is identical to the loss-augmented prediction step that we have already encountered in subgradient-based training. Consequently, the cutting plane method is a prediction-based parameter learning technique.

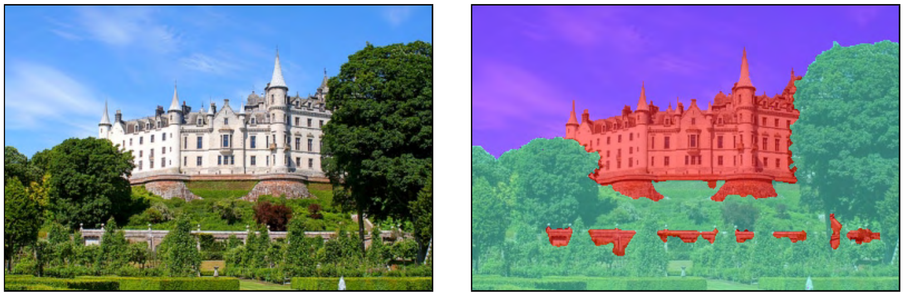


Fig. 6.5 Illustration of *Semantic Image Segmentation*. Each pixel of an image (left) is assigned a semantic class label, here: *sky (blue), building (red), or vegetation (green)*. Spatial consistency of the labeling is encouraged by pair-wise energy terms between neighboring pixels. (Image source: `http://www.flickr.com/photos/foxypar4/3313167875/`).

Example 6.3 (Semantic Image Segmentation). In semantic image segmentation each pixel of an image has to be assigned to one of multiple predefined categories, e.g., material classes ("grass," "road") or geometric properties ("vertical surface," "ground plane"), see Figure 6.5 for a visualization. To solve this task, Szummer et al. [146] use a structured support vector machine to predict a label for every pixel. The unary terms consist of low-level image feature, in particular color and image gradient histograms. The pairwise terms are Potts potentials, measuring if neighboring pixels belong to the same or to different labels, and the corresponding weight is restricted to positive values. By restricting the pairwise weights to positive values, prediction and loss-augmented prediction can be done with the graph cuts algorithm. This allows working set training of the S-SVM.

Cutting plane training is attractive because it allows us to reuse existing components: ordinary SVM solvers and algorithms for (loss-adapted) MAP prediction. However, its convergence rate can be unsatisfactory, in particular for large values of the regularization constant C (see, e.g., [113] for a qualitative as well as quantitative study). The recently developed *one-slack* formulation of S-SVM reduce this problem.

Definition 6.3 (S-SVM — One Slack Formulation). Let $\xi \in \mathbb{R}_+$ be a single auxiliary (slack) variable. For any $C > 0$ *one-slack S-SVM* *training* chooses the parameter w^* by solving

$$(w^*,\xi^*) = \operatorname*{argmin}_{w\in\mathbb{R}^D,\xi\in\mathbb{R}_+} \quad \frac{1}{2}\|w\|^2 + C\xi \tag{6.9}$$

subject to, for all $(y^{(1)},\dots,y^{(n)}) \in \mathcal{Y}\times\cdots\times\mathcal{Y}$,

$$\frac{1}{N}\sum_{n=1}^{N}[g(x^n,y^n,w) - g(x^n,y^{(n)},w)] \geq \frac{1}{N}\sum_{n=1}^{N}\Delta(y^n,y^{(n)}) - \xi. \tag{6.10}$$

S-SVM training with one slack variable is equivalent to S-SVM training with n slack variables and therefore also to the loss formulation of S-SVM training. In fact, it is easy to see that from every solution to (6.7)/(6.8) we obtain a solution to (6.9)/(6.10) by setting $\xi = \sum_{n=1}^{N}\xi^n$. The opposite direction requires a more careful analysis of independences between the constraints, see [63]. The one-slack formulation of S-SVM training has $|\mathcal{Y}|^n$ constraints, so even more than the variant with n slack variables. However, it can be shown that cutting plane optimization of (6.9)/(6.10) achieves a solution ε-close to the optimal objective value within $O(\frac{1}{\varepsilon})$ steps, thereby often offering a significant reduction in training time for practical problems. The intuitive reason for this is that the one-slack formulation builds each of the constraints in its working set as a sum over the feature vectors of several candidate predictions. This leads to a smaller number of overall constraints, while at the same time each constaints contains information from many samples and is therefore more robust.

An equivalent approach for achieving $O(\frac{1}{\varepsilon})$ convergence of S-SVM training is the BMRM procedure [149]. It relies on *bundle methods* to stabilize the objective function between iterations, thereby also achieving faster convergence than a straight-forward cutting plane scheme.

Example 6.4 (Remote Imaging Ground Survey). *Remote imaging ground surveys* aim at classifying each pixel of an aerial or satellite by its surface type, e.g., *roads*, *residential areas*, or *commercial areas*, see Figure 6.6 for an illustration.

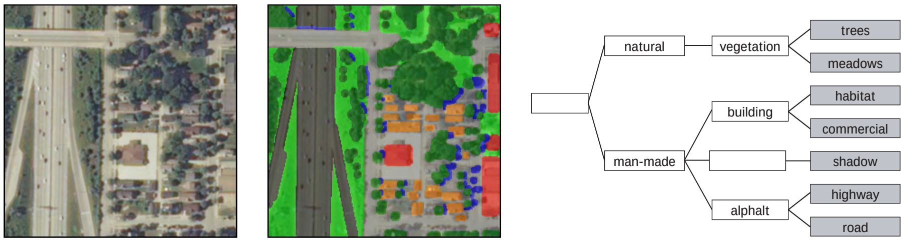


Fig. 6.6 Illustration of automatic remote imaging ground survey [153]. Each pixel of a multispectral aerial image (left) is classified into one class of a seven class hierarchy (right). Outputs (middle) are color coded: *trees (dark green), meadows (light green), highway (dark gray), road (light gray), residential (orange), commercial (red)*, and *shadow (blue)*. (Image Source: WisconsinView.org, SSEC).

Tuia et al. [153] introduce a hierarchical S-SVM classification model for this purpose in which a per-pixel classifier is trained using working set training for the one-slack S-SVM formulation. Because the number of classes is small, prediction and loss-augmented prediction can be performed by exhaustive evaluation.

6.3 Kernelization

Support vector machines derive a large part of their popularity from the fact that they can be *kernelized*, thereby allowing the efficient training of non-linear classifiers. Structured SVMs can be *kernelized* in a similar way.

Definition 6.4 (Kernelized S-SVM). Let $k\colon(\mathcal{X}\times\mathcal{Y})\times(\mathcal{X}\times\mathcal{Y})\to\mathbb{R}$ be a *positive definite joint kernel function* with induced feature function $\varphi\colon\mathcal{X}\times\mathcal{Y}\to\mathcal{H}$ into a Hilbert space $\mathcal{H}$. For any $C>0$, *kernelized S-SVM training* forms a decision function

$$f(x)=\operatorname*{argmax}_{y\in\mathcal{Y}}\sum_{i=1}^{N}\sum_{y'\in\mathcal{Y}}\alpha_{iy'}k(\,(x^i,y'),(x,y)\,),\qquad(6.11)$$

with coefficients $\alpha=(\alpha_{iy})_{i=1,\dots,N,\,y\in\mathcal{Y}}$ that are determined by solving

$$\alpha=\operatorname*{argmax}_{\alpha\in\mathbb{R}_+^{N\times\mathcal{Y}}}\sum_{\substack{i=1,\dots,N\\ y\in\mathcal{Y}}}\alpha_{iy}-\frac{1}{2}\sum_{\substack{i=1,\dots,N\\ y\in\mathcal{Y}}}\sum_{\substack{i'=1,\dots,N\\ y'\in\mathcal{Y}}}\alpha_{iy}\alpha_{i'y'}\overline{K}^{ii'}_{yy'}\qquad(6.12)$$

subject to, for $i = 1,\dots,N$,

$$\sum_{y\in\mathcal{Y}} \alpha_{iy} \le \frac{C}{N}, \tag{6.13}$$

where $\overline{K}^{ii'}_{yy'} = K^{ii'}_{y^i y^{i'}} - K^{ii'}_{y^i y'} - K^{ii'}_{y y^{i'}} + K^{ii'}_{yy'}$ with $K^{ii'}_{yy'} = k((x^i,y),(x^{i'},y'))$.

The kernelized S-SVM formulation is derived by applying the formalism of Lagrangian dualization to the constrained optimization problem (6.7)/(6.8), resulting in its *dual* problem

$$\max_{\alpha\in\mathbb{R}_+^{N\times\mathcal{Y}}} \sum_{\substack{i=1,\dots,N\\ y\in\mathcal{Y}}} \alpha_{iy} - \frac{1}{2} \sum_{\substack{i,i'=1,\dots,N\\ y,y'\in\mathcal{Y}}} \alpha_{iy}\alpha_{i'y'}\langle\varphi(x^i,y^i) - \varphi(x^i,y),\varphi(x^{i'},y^{i'}) - \varphi(x^{i'},y')\rangle, \tag{6.14}$$

subject to, for $i = 1,\dots,N$,

$$\sum_{y\in\mathcal{Y}} \alpha_{iy} \le \frac{C}{N}. \tag{6.15}$$

Using the kernel trick [135] we can replace inner products between feature functions by evaluations of the kernel function

$$k((x,y),(x',y')) = \langle\varphi(x,y),\varphi(x',y')\rangle. \tag{6.16}$$

The optimization problem (6.12)/(6.13) follows from this using the bilinearity of the inner product and the definition of $\overline{K}$.

Note that in principle the prediction function (6.11) might become infeasible to compute, because it contains a potentially exponential number of summands. However, this is not the case in practice because the constraints (6.15) enforce sparsity in the coefficients. For every $i = 1,\dots,N$ most coefficients α_{iy} for $y\in\mathcal{Y}$ will be zero. This sparsity property also makes it feasible to solve the optimization problem (6.12)/(6.13) numerically by keeping a working set over the non-zero coefficients, see [151].

Kernel functions of the form (6.16) are called *joint kernels*, because they are not only kernels between two elements of the input domain, as it is the case for ordinary support vector machines, but between two $(\mathit{input}, \mathit{output})$ pairs. In their kernelized form structured SVMs offer the same advantages of kernelized training as ordinary SVMs do. In particular, one does not need an explicit expression for the feature map φ. It suffices if we can evaluate the kernel function for arbitrary arguments. This is specifically advantageous if the feature map is very high dimensional. However, when choosing a joint kernel function one has to take care that the maximization (6.11) and the loss-augmented prediction during working set training remain feasible.

Example 6.5 (Object Category Localization). The task of localizing object categories in images typically requires the prediction of a bounding box for each object in an image. Blaschko and Lampert [21] construct a kernelized S-SVM for this task that learns a prediction function into the set of 4-tuples of bounding box coordinates.

For this purpose they introduce the *restriction kernel*: $k((x,y), (x',y')) = k_{\text{image}}(x|_y, x'|_{y'})$, where $x|_y$ denotes the restriction of the image x to the rectangular region described by y, and k_{image} is any kernel between images, see Figure (6.7) for an illustration. When k_{image} is a linear kernel over a bag of visual words representation, prediction

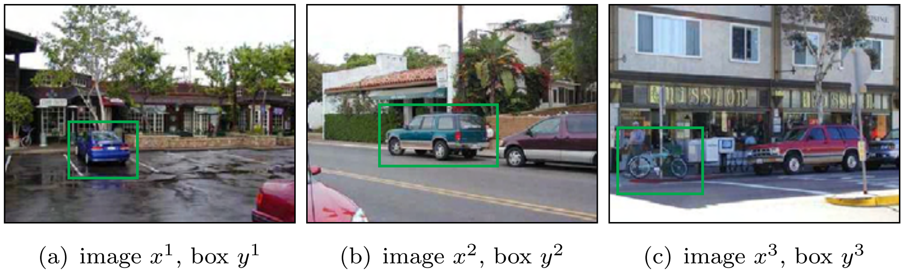

(a) image x^1, box y^1 (b) image x^2, box y^2 (c) image x^3, box y^3

Fig. 6.7 *Restriction kernel* $k((x,y),(x',y')) = k_{\text{image}}(x|_y, x'|_{y'})$ for bounding box object category localization. Image-box pairs are compared by the similarity of the image regions inside the box coordinates. $k((x^1,y^1),(x^2,y^2))$ is large, because the regions show similar objects. $k((x^1,y^1),(x^3,y^3))$ is small, because the objects within the regions are not similar. (Image source: `http://pdphoto.org`)

and loss-augmented prediction are performed efficiently by a branch-and-bound search [90].

6.4 Latent Variables

As we discussed in *Conditional Random Fields*, the introduction of latent variables can give more expressive power to structured prediction models, thereby making them more powerful prediction tools. Like training methods for CRFs, S-SVMs training can be extended to include latent variables in a straight-forward way.

Definition 6.5 (Latent-Variable S-SVMs). Let $z \in \mathcal{Z}$ be a vector of latent variables, i.e., its values are not observed in the training set. Let $\varphi: \mathcal{X} \times \mathcal{Y} \times \mathcal{Z} \to \mathbb{R}^D$ be a feature map, where $\mathcal{Z}$ denotes a set of latent variables that are observed neither at training nor at evaluation time. *Latent variable S-SVM training* [172] learns a prediction function

$$f(x) = \operatorname*{argmax}_{y \in \mathcal{Y}} \max_{z \in \mathcal{Z}} g(x,y,z,w^*) \tag{6.17}$$

with $g(x,y,z,w) = \langle w, \varphi(x,y,z) \rangle$, where the parameter vector is obtained by solving

$$w^* = \operatorname*{argmax}_{w \in \mathbb{R}^D} \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} \ell(x^n,y^n,w) \tag{6.18}$$

with

$$\begin{aligned}\ell(x^n,y^n,w) = {} & \max_{y \in \mathcal{Y}}[\Delta(y^n,y) + \max_{z \in \mathcal{Z}} g(x^n,y,z,w)] \\ & - \max_{z \in \mathcal{Z}} g(x^n,y^n,z,w).\end{aligned} \tag{6.19}$$

Latent variable S-SVM training is derived from the loss-based S-SVM formulation (6.2), by introducing additional latent variables $z \in \mathcal{Z}$ and bounding the Δ-loss by the Hinge-loss when choosing the *best possible assignment*, i.e., when maximizing $g(x,y,z,w)$ over $z \in \mathcal{Z}$.

Because of the max operation in a term with negative sign within Equation (6.19), this upper bound is no longer a convex function of w, such that (6.18) is not a convex optimization problem. We can therefore not expect to find an efficient training procedure with guaranteed convergence to the globally best weight vector. However, good results have been reported using locally optimal optimization techniques, in particular the *concave–convex procedure (CCCP)* [175] for which pseudo-code is given in Algorithm 16.

CCCP is based on the observation that, while not convex, Equation (6.18) can be written as a sum of a concave and a convex term. Both terms have have the form of (loss-augmented) prediction tasks, so individually we can optimize them efficiently. The algorithm works by iteratively approximating the concave part by upper bounding linear functions. This requires the identification of assignments to the latent variables that minimize the concave expression, or equivalently maximize its negative (line 11). Intuitively, the step can be thought of a

Algorithm 16: Concave–Convex Procedure

```
1:  $w^* = \textsc{ConvexConcave}(\varepsilon)$
2:  Input:
3:      $\varepsilon$ precision
4:  Output:
5:      $w^* \in \mathbb{R}^D$ learned weight vector
6:  Algorithm:
7:  $w_{\text{prev}} \leftarrow 0$
8:  repeat
9:    $w_{\text{cur}} \leftarrow w_{\text{prev}}$
10:   for $n = 1, \dots, N$ do
11:     $\hat{z}^n \leftarrow \max_{z \in \mathcal{Z}} g(x^n, y^n, z, w_{\text{cur}})$
12:   end for
13:   $w_{\text{cur}} \leftarrow \text{argmin}_{w \in \mathbb{R}^D} \ \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} \ \hat{\ell}^n(w)$ with
      $\hat{\ell}^n(w) = \max_{y \in \mathcal{Y}, z \in \mathcal{Z}} [\Delta(y^n, y) + g(x^n, y, z, w)] - g(x^n, y^n, \hat{z}^n, w).$
14: until $|w_{\text{prev}} - w_{\text{cur}}| < \varepsilon$
15: $w^* \leftarrow w_{\text{cur}}$
```

finding the latent assignments that best explain the training data for the current weight vector and fixing them for the rest of the iteration. Subsequently, one solves the modified optimization problem (line 13), which is now convex, because the concave part has been linearized, and one obtains a new weight vector. Both steps are iterated until no improvement in the objective value is achieved anymore.

The CCCP procedure decreases the objective value at each iteration, but it does not guarantee that the solution found at convergence is the global minimum. Nevertheless, CCCP has shown useful in practice for solving latent variable problems.

Example 6.6(Occlusion-Aware Object Category Localization). Vedaldi and Zisserman [156] extend S-SVM based object category localization (Example 6.5) to better cope with partially occluded or truncated objects. For each images x with bounding box y they introduce a vector of hidden binary variables z that encodes which parts of the object is visible. For this they decompose the object region $x|_y$ into rectangular grid cells. Each component of z corresponds to one such grid cell, indicating if it is occluded or not (see Figure 6.8 for an illustration). Prediction and loss-augmented prediction can be performed efficiently by incorporating idea from multiscale template matching. This is used to train the model with CCCP.

6.5 Other Training Objectives

Prediction-based training methods for structured prediction models are powerful because they only require iterative solution of the loss-augmented prediction problem, and this is often possible using the same routines as for MAP prediction. However, there are cases in which the loss function cannot easily be included into the optimization. When loss-augmented prediction becomes a hard problem exact structured SVM training typically becomes infeasible. Assuming that ordinary, loss-free, prediction is still possible, the *structured perceptron* offers a solution in these cases.

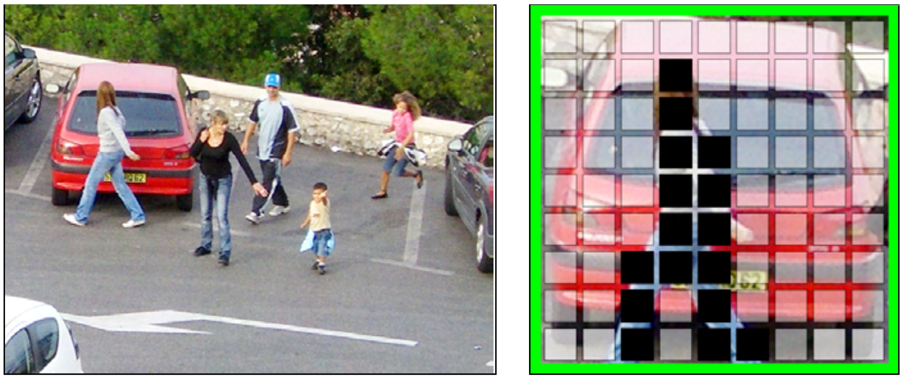


Fig. 6.8 Illustration of Object Category Localization with latent occlusion variables (simplified from Vedaldi and Zisserman [156]). Left: as in ordinary object category localization, we aim at learning a prediction function that takes images as input and bounding box coordinates as output. Right: the presumed object region is decomposed into a rectangular grid (here 9 × 9). For each grid cell a latent variable indicates whether the object is visible (transparent) or occluded (black).

6.5.1 Structured Perceptron Training

Structured perceptron learning [33] generalizes the multi-class perceptron [35] to structured output spaces. Algorithm 17 shows it in pseudo-code. Like a classical perceptron, the structured perceptron works iteratively. In every step it chooses a training examples, either randomly or in an online fashion, and predicts its label (line 10). If the prediction was correct, the weight vector is kept, otherwise the feature vector of the correct prediction is added and the feature vector of the wrong prediction is subtracted (line 12). Thereby the quality estimate of the correct label is increased for the next step and the estimate of the mispredicted label is decreased. As one can see, no loss function is taken into account, and only ordinary prediction is required. The structured perceptron is also easily adapted to the situation of online-learning, as it only requires access to one training example at a time.

Because Algorithm 17 does not include explicit regularization, overfitting can occur and structured perceptron learning typically results in lower prediction accuracy than S-SVM training when both methods are applicable. This holds in weaker form also for variants such as the *averaged structured perceptron* which introduces regularization

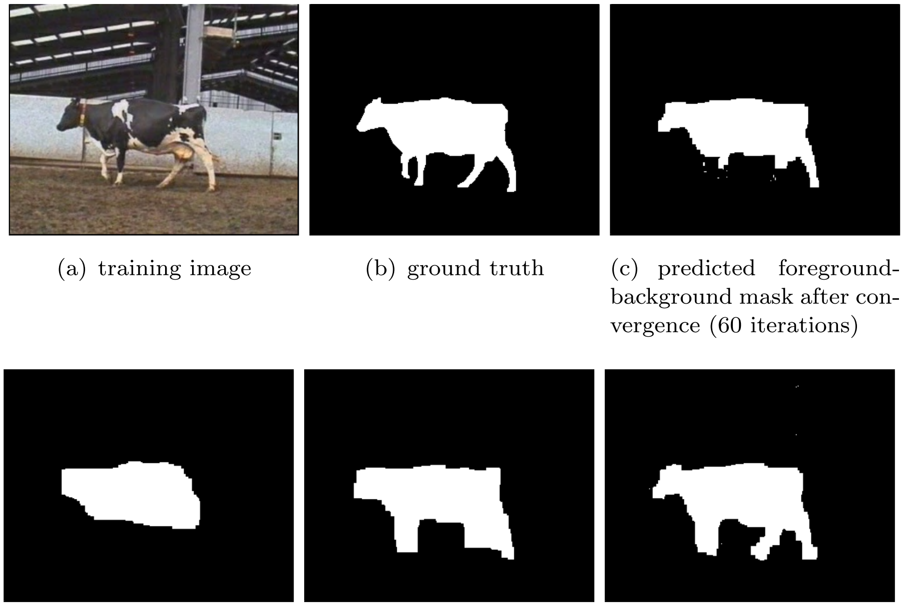
Algorithm 17: Structured Perceptron Training

```
$w^* = \textsc{StructuredPerceptron}(T)$
Input:
    $T$ number of iterations
Output:
    $w^* \in \mathbb{R}^D$ learned weight vector
Algorithm:
$w_{\text{cur}} \leftarrow 0$
for $t = 1,\dots,T$ do
    $(x^n, y^n) \leftarrow$ randomly chosen training example
    $\hat{y} \leftarrow \text{argmax}_{y \in \mathcal{Y}} \langle w_{\text{cur}}, \varphi(x^n, y^n) \rangle$
    if $\hat{y} \neq y^n$ then
        $w_{\text{cur}} \leftarrow w_{\text{cur}} + \big[\varphi(x^n, y^n) - \varphi(x^n, \hat{y})\big]$
    end if
end for
$w^* \leftarrow w_{\text{cur}}$
```

by returning the average of all weight vector obtained during training instead of the final one [33].

Perceptron-based structured learning nevertheless has its justification due to its extreme simplicity in implementation. Used as a baseline, it provides insight into the learning problem as well as providing a good indicator what results to expect from more advanced techniques for the learning of structured prediction functions.

Example 6.7 (Figure-Ground Image Segmentation). Figure-ground segmentation (see Example 5.1) can also be performed using prediction-based learning. Structured perceptron learning leads to a particularly simple training algorithm that is illustrated in Figure 6.9. In each step, one picks a training image and predicts the segmentation resulting from the current weight vector, e.g., using the graph cuts algorithm. If its overlap with the ground truth is too low, the weight vector is updated by the difference of feature representations between the target and the predicted segmenetation. Otherwise the weight vector



(a) training image

(b) ground truth

(c) predicted foreground-background mask after convergence (60 iterations)

(d) predicted masks after 1, 10 and 50 structured perceptron iterations

Fig. 6.9 Illustration of prediction-based learning for *figure-ground segmentation*. Each pixel of an image has to be label foreground (white) or background (black). Structured perceptron learning predicts a segmentation in each iteration and changes the weight vector until for all training images the prediction is sufficiently close to the ground truth segmentation. In this example, the criterion is 90% area overlap which is achieved after 60 iterations. (Image source: Derek Magee, University of Leeds.).

is left unchanged. The training is stopped when the predictions for all examples are sufficiently good, or after a predefined number of steps.

6.6 Approximate Training

Prediction-based training generally assumes that the prediction problem $\text{argmax}_y\, g(x,y,w)$ or the loss-augmented prediction problem $\text{argmax}_y\, \Delta(y',y) + g(x,y,w)$ can be solved exactly. As we saw in *Structured Prediction* this is generally not the case, and we often have to *give up optimality* to be able to predict a structured output at all.

It is currently a field of active research how prediction-based training and approximate prediction techniques can be combined. Early theoretical results by Kulesza and Pereira [86] showed the severity of the problem: when training a structured perceptron already small prediction errors can lead to large deviations of the learned weight vector. However, the authors also identified subclasses of problems for which this problem does not occur.

Finley and Joachims [39] analyzed the situation of S-SVMs in more detail, when the loss-augment prediction problem can be performed only approximately. They identified two classes of approximate inference techniques: *under-generating* techniques, such as greedy search or the loopy max-product algorithm, make the intractable prediction problem tractable by searching only over a subset $\mathcal{Y}_{\text{under}} \subset \mathcal{Y}$ of possible labels. *Over-generating* techniques achieve the same goal by searching over a set $\mathcal{Y}_{\text{over}} \supset \mathcal{Y}$ that is larger than the original output space, e.g., real values outputs instead of integer valued ones for linear programming relaxations.

Both classes have advantages and disadvantages, but in conclusion over-generating techniques appear more suitable for approximate S-SVM training than under-generating ones, which was confirmed by experimental results.

Alternatively, training modification to the S-SVM training have been proposed that approximate the optimization problem in a way similar to pseudo-likelihood training [22], or to piece-wise training of CRFs [4]. It has also been suggested to train hybrid generative/ discriminative models that avoid the need to perform prediction tasks at training time altogether [88].

So far practical experience with approximate training of structured models in computer vision tasks is limited, and it appears that more research is required to determine which direction is the most promising one.

7

Conclusion

Making use of the problem structure is crucial for efficient structured prediction and learning. The models we discuss in this monograph all have a nontrivial structure; this makes them rich enough to be useful, but it also complicates inference and parameter estimation.

Many of the occurring prediction problems are NP-hard. Only by understanding the problems' structure, we are able to still make predictions, and do so efficiently. To this end we discussed a number of successful techniques in terms of what desired property of a nonexisting *ideal algorithm* we are willing to give up: problem generality, guaranteed optimality, worst-case complexity, integrality, and determinism.

Parameter learning poses very similar problems as prediction does, no matter if we follow a probabilistic or a maximum-margin approach. Understanding the model structure is important for accessing the insights obtained by solving the prediction problem. This allows us to make the right choices that make training tractable.

Despite the breadth of the algorithmic techniques described and the emergence of general frameworks such as graphical models, it is ultimately the individual computer vision researcher who has to face a novel structured prediction problem.

Notations and Acronyms

Symbol	Description
$\mathcal{X}$	Input domain of the prediction function.
$x \in \mathcal{X}$	Element of input domain.
X	Random variable that takes values in input domain.
$\mathcal{G}$	Decision domain.
$\mathcal{Y} \subseteq \mathcal{G}$	Structured output domain of the prediction function.
$y \in \mathcal{Y}$	Element of output domain.
Y	Random variable that takes values in output domain.
$\mathcal{Z}$	Domain of latent values.
$z \in \mathcal{Z}$	Latent value.
Z	Latent random variable.
$G = (V,\mathcal{E})$	Graph for graphical model. $V = \{\textit{nodes}\}$, $\mathcal{E} = \{\textit{edges}\}$.
$(V,\mathcal{F},\mathcal{E})$	Factor graph. $V = \{\textit{variable nodes}\}$, $\mathcal{F} = \{\textit{factor nodes}\}$, $\mathcal{E} = \{\textit{edges}\}$.
$\Box_i$	Part of structured object $\Box$ that corresponds to node $i \in V$ in graphical model representation, in particular $\mathcal{X}_i$, X_i, x_i for inputs, $\mathcal{Y}_i, Y_i, y_i$ for outputs, and $\mathcal{Z}_i, Z_i, z_i$ for latent quantities.
$\Box_F$	Part of structured object $\Box$ that corresponds to factor F in graphical model representation, in particular $\mathcal{X}_F = \prod_{i \in F} \mathcal{X}_i$, $X_F = (X_i)_{i \in F}$, $x_F = (x_i)_{i \in F}$, etc.
$\Box^*$	Result of MAP-prediction for a quantity $\Box$
$\hat{\Box}$	Empirical estimate of a quantity $\Box$.

(*Continued*)

Symbol	Description
$D \in \mathbb{N}$	Dimension of feature space.
$\varphi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^D$	Feature function.
$w \in \mathbb{R}^D$	weight vector.
$g : \mathcal{X} \times \mathcal{G} \to \mathbb{R}$	Evaluation function, usually parametrized as $g(x,y,w) = \langle w, \varphi(x,y)\rangle$.
$f : \mathcal{X} \to \mathcal{Y}$	Structured prediction function, usually defined implicitly as $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} g(x,y,w)$.
$p(Y = y \mid X = x, W = w)$	Conditional probability distribution parametrized by w.
$E(x,y,w)$	Energy function, $E(x,y,w) = -\log p(y\mid x,w)$.
$\mathcal{S}$	Set of samples.
$\tau^{(k)} > 0$	Simulated annealing temperature at iteration k.
$\eta^{(k)} \in \mathbb{N}$	Simulation steps in simulated annealing iteration k.
$\mathcal{D} = \{(x^i, y^i)\}_{i=1,\dots,N}$	Set of *input-output* pairs used for training
$N \in \mathbb{N}$	Number of training examples
$T \in \mathbb{N}$	Number of training iterations
$d \in \mathbb{R}^D$	Descent direction in gradient descent procedures.
$\eta_1, \dots, \eta_T$	Sequence of learning rates, typically $\eta_t = \frac{\eta}{t}$ for $\eta > 0$.
$\langle \cdot, \cdot \rangle$	(Euclidean) inner product: $\langle u, v\rangle = u^t v = \sum_{i=1}^{dim} [u]_i [v]_i$
$[\![\cdot]\!]$	Function evaluating to 1 if its argument is true, to 0 otherwise.
$[\cdot]_+$	Positive part of argument: $[t]_+ := \max(0, t)$.

References

[1] E. H. L. Aarts, J. H. M. Korst, and P. J. M. v. Laarhoven, "Simulated annealing," in *Local Search in Combinatorial Optimization*, (E. H. L. Aarts and J. K. Lenstra, eds.), pp. 91–120, Wiley-Interscience, 1997.

[2] R. K. Ahuja, Ö. Ergun, J. B. Orlin, and A. P. Punnen, "A survey of very large-scale neighborhood search techniques," in *Workshop on Discrete Optimization*, (E. Boros and P. L. Hammer, eds.), pp. 75–102, 2002.

[3] K. Alahari, P. Kohli, and P. H. S. Torr, "Reduce, reuse & recycle: Efficiently solving multi-label MRFs," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[4] K. Alahari, C. Russell, and P. H. S. Torr, "Efficient piecewise learning for conditional random fields," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[5] M. Andriluka, S. Roth, and B. Schiele, "Pictorial structures revisited: People detection and articulated pose estimation," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[6] K. M. Anstreicher and L. A. Wolsey, "Two "well-known" properties of subgradient optimization," *Mathematical Programming*, vol. 120, no. B, pp. 213–220, 2009.

[7] A. U. Asuncion, Q. Liu, A. T. Ihler, and P. Smyth, "Learning with blocks: Composite likelihood and contrastive divergence," in *Conference on Uncertainty in Artificial Intelligence (AISTATS)*, 2010.

[8] F. Barahona and R. Anbil, "The volume algorithm: Producing primal solutions with a subgradient method," *Mathematical Programming*, vol. 87, no. 3, pp. 385–399, 2000.

[9] D. Barber, *Bayesian Reasoning and Machine Learning.* Cambridge University Press, 2011. In press.

[10] A. Barbu and S. C. Zhu, “Generalizing swendsen-wang to sampling arbitrary posterior probabilities,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, vol. 27, no. 8, pp. 1239–1253, 2005.

[11] D. Batra, A. C. Gallagher, D. Parikh, and T. Chen, “Beyond trees: MRF inference via outer-planar decomposition,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[12] D. Batra, S. Nowozin, and P. Kohli, “Tighter relaxations for MAP-MRF inference: A local primal-dual gap based separation algorithm,” in *Conference on Uncertainty in Artificial Intelligence (AISTATS)*, 2011.

[13] D. P. Bertsekas, *Nonlinear Programming.* Belmont, MA: Athena Scientific, 2nd Edition, 1995.

[14] D. P. Bertsekas, *Network Optimization.* Athena Scientific, 1998.

[15] D. Bertsimas and J. N. Tsitsiklis, *Introduction to Linear Optimization.* Athena Scientific, 1997.

[16] J. Besag, “Statistical analysis of non-lattice data,” *The Statistician*, pp. 179–195, 1975.

[17] J. Besag, “On the statistical analysis of dirty pictures,” *Journal of the Royal Statistical Society*, vol. B-48, no. 3, pp. 259–302, 1986.

[18] S. Birchfield and C. Tomasi, “A pixel dissimilarity measure that is insensitive to image sampling,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, vol. 20, no. 4, pp. 401–406, 1998.

[19] C. M. Bishop, *Pattern recognition and machine learning.* Springer, 2006.

[20] A. Blake, C. Rother, M. Brown, P. Perez, and P. H. S. Torr, “Interactive image segmentation using an adaptive GMMRF model,” in *European Conference on Computer Vision (ECCV)*, pp. 428–441, 2004.

[21] M. B. Blaschko and C. H. Lampert, “Learning to localize objects with structured output regression,” in *European Conference on Computer Vision (ECCV)*, Springer, 2008.

[22] L. Bo and C. Sminchisescu, “Structured output-associative regression,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[23] J. F. Bonnans, J. C. Gilbert, C. Lemaréchal, and C. A. Sagastizábal, *Numerical Optimization.* Springer, 2003.

[24] A. Bordes, L. Bottou, and P. Gallinari, “SGD-QN: Careful Quasi-Newton stochastic gradient descent,” *Journal of Machine Learning Research (JMLR)*, vol. 10, pp. 1737–1754, 2009.

[25] L. Bottou and O. Bousquet, “The tradeoffs of large scale learning,” in *Conference on Neural Information Processing Systems (NIPS)*, The MIT Press, 2007.

[26] Y. Boykov and V. Kolmogorov, “An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision,” *PAMI*, vol. 26, no. 9, pp. 1124–1137, 2004.

[27] Y. Boykov, O. Veksler, and R. Zabih, “Markov Random Fields with Efficient Approximations,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 648–655, IEEE Computer Society, 1998.

[28] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, vol. 23, no. 11, pp. 1222–1239, 2001.

[29] Y. Boykov and M.-P. Jolly, "Interactive graph cuts for optimal boundary and region segmentation of objects in N-D images," in *International Conference on Computer Vision (ICCV)*, pp. 105–112, 2001.

[30] J. Carreira and C. Sminchisescu, "Constrained parametric min-cuts for automatic object segmentation," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3241–3248, 2010.

[31] C. Chen, D. Freedman, and C. H. Lampert, "Enforcing topological constraints in random field image segmentation," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2089–2096, 2011.

[32] J. Clausen, *Branch and Bound Algorithms — Principles and Examples.* University of Copenhagen, 1999.

[33] M. Collins, "Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms," in *Conference on Empirical methods in Natural Language Processing*, pp. 1–8, 2002.

[34] A. J. Conejo, E. Castillo, R. Mínguez, and R. García-Bertrand, *Decomposition Techniques in Mathematical Programming.* Springer, 2006.

[35] K. Crammer and Y. Singer, "Ultraconservative online algorithms for multiclass problems," *Journal of Machine Learning Research (JMLR)*, vol. 3, pp. 951–991, 2003.

[36] G. Elidan, I. McGraw, and D. Koller, "Residual belief propagation: Informed scheduling for asynchronous message passing," in *Uncertainty in Artificial Intelligence (UAI)*, 2006.

[37] P. Felzenszwalb and D. Huttenlocher, "Efficient matching of pictorial structures," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 66–75, 2000.

[38] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient belief propagation for early vision," *International Journal of Computer Vision (IJCV)*, vol. 70, no. 1, pp. 41–54, 2006.

[39] T. Finley and T. Joachims, "Training structural SVMs when exact inference is intractable," in *International Conference on Machine Learing (ICML)*, pp. 304–311, 2008.

[40] M. A. Fischler and R. A. Elschlager, "The representation and matching of pictorial structures," *IEEE Trans. Computer*, vol. 22, no. 1, pp. 67–92, January 1973.

[41] R. Fletcher, *Practical Methods of Optimization.* John Wiley & Sons, 1987.

[42] V. Franc, S. Sonnenburg, and T. Werner, "Cutting-plane methods in machine learning," in *Optimization for Machine Learning*, (S. Sra, S. Nowozin, and S. J. Wright, eds.), MIT Press, 2011.

[43] A. Frangioni, "About lagrangian methods in integer optimization," *Annals of Operations Research*, vol. 139, no. 1, pp. 163–193, 2005.

[44] D. Freedman and P. Drineas, "Energy minimization via graph cuts: Settling what is possible," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 939–946, 2005.

[45] W. T. Freeman, E. C. Pasztor, and O. T. Carmichael, “Learning low-level vision,” *International Journal of Computer Vision (IJCV)*, vol. 40, no. 1, pp. 25–47, 2000.

[46] B. J. Frey and D. J. C. MacKay, “A revolution: Belief propagation in graphs with cycles,” in *Conference on Neural Information Processing Systems (NIPS)*, The MIT Press, 1997.

[47] B. Fulkerson, A. Vedaldi, and S. Soatto, “Class segmentation and object localization with superpixel neighborhoods,” in *International Conference on Computer Vision (ICCV)*, 2009.

[48] D. Geiger and A. L. Yuille, “A common framework for image segmentation,” *International Journal of Computer Vision (IJCV)*, vol. 6, no. 3, pp. 227–243, 1991.

[49] S. Geman and D. Geman, “Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, vol. 6, no. 6, pp. 721–741, 1984.

[50] A. M. Geoffrion, “Lagrangian relaxation for integer programming,” *Mathematical Programming Study*, vol. 2, pp. 82–114, 1974.

[51] C. J. Geyer, “Practical Markov chain Monte Carlo,” *Statistical Science*, vol. 7, no. 4, pp. 473–483, 1992.

[52] J. Goodman, “Exponential priors for maximum entropy models,” in *Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, pp. 305–312, 2004.

[53] M. Guignard, “Lagrangean relaxation,” *TOP*, vol. 11, no. 2, pp. 151–200, 2003.

[54] M. Guignard and S. Kim, “Lagrangean decomposition: A model yielding stronger Lagrangean bounds,” *Mathematical Programming*, vol. 39, pp. 215–228, 1987.

[55] O. Häggström, *Finite Markov Chains and Algorithmic Applications*. Cambridge University Press, 2000.

[56] P. Hart, N. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[57] W. K. Hastings, “Monte Carlo sampling methods using Markov chains and their applications,” *Biometrika*, pp. 97–109, 1970.

[58] X. He, R. Zemel, and M. Carreira-Perpinan, “Multiscale conditional random fields for image labeling,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004.

[59] T. Heskes, “Convexity arguments for efficient minimization of the Bethe and Kikuchi free energies,” *Journal of Artificial Intelligence Research (JAIR)*, vol. 26, pp. 153–190, 2006.

[60] M. R. Hestenes and E. Stiefel, “Methods of conjugate gradients for solving linear systems,” *Journal of Research of the National Bureau of Standards*, vol. 49, no. 6, pp. 409–436, 1952.

[61] G. Hinton, “Training products of experts by minimizing contrastive divergence,” *Neural Computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

[62] H. Ishikawa, "Exact optimization for Markov random fields with convex priors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 10, pp. 1333–1336, 2003.

[63] T. Joachims, T. Finley, and C.-N. Yu, "Cutting-plane training of structural SVMs," *Machine Learning*, vol. 77, no. 1, pp. 27–59, 2009.

[64] J. K. Johnson, D. M. Malioutov, and A. S. Willsky, "Lagrangian relaxation for MAP estimation in graphical models," *Allerton Conference on Control, Communication and Computing*, 2007.

[65] V. Jojic, S. Gould, and D. Koller, "Accelerated dual decomposition for MAP inference," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21–24, 2010, Haifa, Israel*, (J. Fürnkranz and T. Joachims, eds.), pp. 503–510, Omnipress, 2010.

[66] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, "An introduction to variational methods for graphical models," *Machine Learning*, vol. 37, no. 2, pp. 183–233, 1999.

[67] J. Kelley Jr, "The cutting-plane method for solving convex programs," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 4, pp. 703–712, 1960.

[68] B. M. Kelm, N. Müller, B. H. Menze, and F. A. Hamprecht, "Bayesian estimation of smooth parameter maps for dynamic contrast-enhanced MR images with block-ICM," in *CVPR Workshop on Mathematical Methods in Biomedical Image Analysis, Computer Vision and Pattern Recognition*, pp. 96–103, 2006.

[69] R. Kikuchi, "A Theory of Cooperative Phenomena," *Physical Review*, vol. 81, no. 6, pp. 988–1003, 1951.

[70] T. Kim, S. Nowozin, P. Kohli, and C. D. Yoo, "Variable grouping for energy minimization," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[71] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[72] J. Kittler and J. Föglein, "Contextual classification of multispectral pixel data," *Image Vision Computing*, vol. 2, no. 1, pp. 13–29, 1984.

[73] P. Kohli, M. P. Kumar, and C. Rother, "MAP inference in discrete models," Tutorial at ICCV 2009, 2009.

[74] P. Kohli, M. P. Kumar, and P. H. S. Torr, "P3 & beyond: Solving energies with higher order cliques," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[75] P. Kohli, L. Ladický, and P. H. S. Torr, "Robust higher order potentials for enforcing label consistency," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[76] P. Kohli, L. Ladicky, and P. H. S. Torr, "Robust higher order potentials for enforcing label consistency," *International Journal of Computer Vision (IJCV)*, vol. 82, no. 3, pp. 302–324, 2009.

[77] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques.* The MIT Press, 2009.

[78] V. Kolmogorov, "Convergent tree-reweighted message passing for energy minimization," *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, vol. 28, no. 10, pp. 1568–1583, 2006.

[79] V. Kolmogorov and C. Rother, "Minimizing nonsubmodular functions with graph cuts-A review," *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, vol. 29, no. 7, pp. 1274–1279, 2007.

[80] V. Kolmogorov and R. Zabih, "What energy functions can be minimized via graph cuts?," *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, vol. 26, no. 2, pp. 147–159, 2004.

[81] N. Komodakis and N. Paragios, "Beyond loose LP-relaxations: Optimizing MRFs by repairing cycles," in *European Conference on Computer Vision (ECCV)*, 2008.

[82] N. Komodakis, N. Paragios, and G. Tziritas, "MRF optimization via dual decomposition: Message-passing revisited," in *International Conference on Computer Vision (ICCV)*, 2007.

[83] N. Komodakis, G. Tziritas, and N. Paragios, "Fast, approximately optimal solutions for single and dynamic MRFs," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE Computer Society, 2007.

[84] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms.* Springer, 4th Edition, 2008.

[85] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498–519, 2001.

[86] A. Kulesza and F. Pereira, "Structured learning with approximate inference," in *Conference on Neural Information Processing Systems (NIPS)*, 2007.

[87] S. Kumar and M. Hebert, "Discriminative fields for modeling spatial dependencies in natural images," *Conference on Neural Information Processing Systems (NIPS)*, 2004.

[88] C. H. Lampert and M. B. Blaschko, "Structured prediction by joint kernel support estimation," *Machine Learning*, vol. 77, no. 2-3, pp. 249–269, 2009.

[89] C. H. Lampert, M. B. Blaschko, and T. Hofmann, "Beyond sliding windows: Object localization by Efficient Subwindow Search," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[90] C. H. Lampert, M. B. Blaschko, and T. Hofmann, "Efficient subwindow search: A branch and bound framework for object localization," *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, vol. 31, no. 12, pp. 2129–2142, 2009.

[91] Y. Lee, Y. Lin, and G. Wahba, "Multicategory support vector machines," *Journal of the American Statistical Association*, vol. 99, no. 465, pp. 67–81, 2004.

[92] C. Lemaréchal, "Lagrangian relaxation," in *Computational Combinatorial Optimization*, (M. Jünger and D. Naddef, eds.), pp. 112–156, Springer, 2001.

[93] V. S. Lempitsky, A. Blake, and C. Rother, "Image segmentation by branch-and-mincut," in *European Conference on Computer Vision (ECCV)*, 2008.

[94] V. S. Lempitsky, P. Kohli, C. Rother, and T. Sharp, "Image segmentation with a bounding box prior," in *International Conference on Computer Vision (ICCV)*, 2009.

[95] F. Liang, C. Liu, and R. J. Carroll, *Advanced Markov Chain Monte Carlo Methods: Learning from Past Samples.* John Wiley, 2010.

[96] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, no. 1, pp. 503–528, 1989.

[97] J. S. Liu, *Monte Carlo Strategies in Scientific Computing, Springer Series in Statistics.* New York: Springer, 2001.

[98] D. J. C. MacKay, *Information Theory, Inference and Learning Algorithms.* Cambridge University Press, 2003.

[99] A. F. T. Martins, N. A. Smith, and E. P. Xing, "Polyhedral outer approximations with application to natural language parsing," in *International Conference on Machine Learing (ICML)*, 2009.

[100] A. F. T. Martins, N. A. Smith, E. P. Xing, P. M. Aguiar, and M. A. T. Figueiredo, "Augmenting dual decomposition for MAP inference," in *Proceedings of the 3rd International Workshop on Optimization for Machine Learning (OPT 2010), December 10, 2010, Whistler, Canada*, 2010.

[101] D. McAllester, "Generalization bounds and consistency for structured labeling," in *Predicting Structured Data*, (G. Bakır, T. Hofmann, B. Schölkopf, A. Smola, B. Taskar, and S. Vishwanathan, eds.), The MIT Press, 2007.

[102] T. Meltzer, A. Globerson, and Y. Weiss, "Convergent message passing algorithms — a unifying view," in *Uncertainty in Artificial Intelligence (UAI)*, 2009.

[103] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, *et al.*, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[104] M. Mézard and A. Montanari, *Information, Physics and Computation.* Oxford University Press, 2009.

[105] T. Minka, "Divergence measures and message passing," Microsoft Research Technical Report, MSR-TR-2005-173, 2005.

[106] L.-P. Morency, A. Quattoni, and T. Darrell, "Latent-dynamic discriminative models for continuous gesture recognition," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[107] G. Mori, "Guiding model search using segmentation," in *International Conference on Computer Vision (ICCV)*, 2005.

[108] I. Murray, "Markov chain Monte Carlo," Tutorial at Machine Learning Summer School 2009, 2009.

[109] A. Nedic and D. Bertsekas, "Convergence rate of incremental subgradient algorithms," in *Stochastic Optimization: Algorithms and Applications*, (S. P. Uryasev and P. M. Pardalos, eds.), pp. 223–264, Springer, 2000.

[110] Y. E. Nesterov, "Smooth minimization of non-smooth functions," *Mathematical Programming*, vol. 103, no. 1, pp. 127–152, 2005.

[111] J. Neter, M. Kutner, C. Nachtsheim, and W. Wasserman, *Applied Linear Statistical Models.* McGraw-Hill, 4 Edition, 1996.

[112] J. Nocedal and S. J. Wright, *Numerical optimization*. Springer, 1999.

[113] S. Nowozin, P. V. Gehler, and C. H. Lampert, “On parameter learning in CRF-based approaches to object class image segmentation,” in *European Conference on Computer Vision (ECCV)*, 2010.

[114] S. Nowozin and S. Jegelka, “Solution stability in linear programming relaxations: graph partitioning and unsupervised learning,” in *International Conference on Machine Learing (ICML)*, 2009.

[115] S. Nowozin and C. H. Lampert, “Global connectivity potentials for random field models,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[116] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: Algorithms and complexity*. Dover Publications, 1998.

[117] J. C. Picard and M. Queyranne, “On the structure of all minimum cuts in a network and applications,” *Combinatorial Optimization II*, pp. 8–16, 1980.

[118] J. C. Platt, “Fast training of support vector machines using sequential minimal optimization,” in *Advances in Kernel Methods*, (B. Schölkopf, C. J. C. Burges, and A. J. Smola, eds.), pp. 185–208, The MIT Press, 1999.

[119] J. C. Platt, “Probabilities for SV Machines,” in *Advances in Large Margin Classifiers*, (A. J. Smola, P. L. Bartlett, B. Schölkopf, and D. Schuurmans, eds.), pp. 61–74, The MIT Press, 1999.

[120] P. Pletscher, C. S. Ong, and J. M. Buhmann, “Entropy and margin maximization for structured output learning,” in *European Conference on Marchine Learning (ECML)*, 2010.

[121] B. Potetz, “Efficient belief propagation for vision using linear constraint nodes,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[122] S. Ramalingam, P. Kohli, K. Alahari, and P. H. S. Torr, “Exact inference in multi-label CRFs with higher order cliques,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[123] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, “Maximum margin planning,” in *International Conference on Machine Learing (ICML)*, 2006.

[124] X. Ren and J. Malik, “Learning a classification model for segmentation,” in *International Conference on Computer Vision (ICCV)*, 2003.

[125] C. P. Robert, *The Bayesian Choice. From Decision Theoretic Foundations to Computational Implementation, Springer Series in Statistics*. Springer, 2001.

[126] C. P. Robert and G. Casella, *Monte Carlo Statistical Methods*. Springer, 2nd Edition, 2004.

[127] C. Rother, V. Kolmogorov, V. S. Lempitsky, and M. Szummer, “Optimizing binary MRFs via extended roof duality,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[128] S. Sarawagi and R. Gupta, “Accurate max-margin training for structured output spaces,” in *International Conference on Machine Learing (ICML)*, 2008.

[129] L. K. Saul and M. I. Jordan, “Exploiting tractable substructures in intractable networks,” in *Conference on Neural Information Processing Systems (NIPS)*, pp. 486–492, 1995.

[130] B. D. Savchynskyy, J. Kappes, S. Schmidt, and C. Schnörr, “A study of Nesterov’s scheme for Lagrangian decomposition and MAP labeling,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE Computer Society, 2011.

[131] M. I. Schlesinger, “Syntactic analysis of two-dimensional visual signals in noisy conditions in Russian,” *Kibernetika*, vol. 4, pp. 113–130, 1976.

[132] F. R. Schmidt, D. Farin, and D. Cremers, “Fast matching of planar shapes in sub-cubic runtime,” in *International Conference on Computer Vision (ICCV)*, 2007.

[133] F. R. Schmidt, E. Töppe, and D. Cremers, “Efficient planar graph cuts with applications in computer vision,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[134] U. Schmidt, Q. Gao, and S. Roth, “A generative perspective on MRFs in low-level vision,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[135] B. Schölkopf and A. J. Smola, *Learning with Kernels.* The MIT Press, 2002.

[136] N. N. Schraudolph and D. Kamenetsky, “Efficient exact inference in planar ising models,” in *Conference on Neural Information Processing Systems (NIPS)*, The MIT Press, 2008.

[137] N. N. Schraudolph and D. Kamenetsky, “Efficient exact inference in planar ising models,” in *Conference on Neural Information Processing Systems (NIPS)*, 2008.

[138] S. E. Shimony, “Finding MAPs for belief networks Is NP-hard,” *Artificial Intelligence*, vol. 68, no. 2, pp. 399–410, 1994.

[139] N. Z. Shor, *Minimization Methods for Non-differentiable Functions.* Springer, 1985.

[140] D. Sontag, A. Globerson, and T. Jaakkola, “Clusters and coarse partitions in LP relaxations,” in *Conference on Neural Information Processing Systems (NIPS)*, 2008.

[141] D. Sontag, A. Globerson, and T. Jaakkola, “Introduction to dual decomposition for inference,” in *Optimization for Machine Learning*, (S. Sra, S. Nowozin, and S. J. Wright, eds.), MIT Press, 2011.

[142] D. Sontag, T. Meltzer, A. Globerson, T. Jaakkola, and Y. Weiss, “Tightening LP Relaxations for MAP using Message Passing,” in *Uncertainty in Artificial Intelligence (UAI)*, pp. 503–510, 2008.

[143] P. Strandmark and F. Kahl, “Parallel and distributed graph cuts by dual decomposition,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[144] C. Sutton and A. McCallum, “Piecewise training for structured prediction,” *Machine Learning*, vol. 77, no. 2–3, pp. 165–194, 2009.

[145] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. F. Tappen, and C. Rother, “A comparative study of energy minimization methods for markov random fields with smoothness-based priors,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, vol. 30, no. 6, pp. 1068–1080, 2008.

[146] M. Szummer, P. Kohli, and D. Hoiem, “Learning CRFs using graph cuts,” in *European Conference on Computer Vision (ECCV)*, 2008.

[147] M. F. Tappen and W. T. Freeman, "Comparison of graph cuts with belief propagation for stereo, using identical MRF parameters," in *International Conference on Computer Vision (ICCV)*, pp. 900–907, IEEE Computer Society, 2003.

[148] B. Taskar, C. Guestrin, and D. Koller, "Max-margin Markov networks," in *Conference on neural information processing systems (NIPS)*, 2003.

[149] C. Teo, S. Vishwanathan, A. Smola, and Q. Le, "Bundle methods for regularized risk minimization," *Journal of Machine Learning Research (JMLR)*, vol. 1, pp. 1–55, 2009.

[150] L. Torresani, V. Kolmogorov, and C. Rother, "Feature correspondence via graph matching: Models and global optimization," in *European Conference on Computer Vision (ECCV)*, 2008.

[151] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, "Large margin methods for structured and interdependent output variables," *Journal of Machine Learning Research (JMLR)*, vol. 6, pp. 1453–1484, 2005.

[152] Z. Tu, X. Chen, A. L. Yuille, and S. C. Zhu, "Image parsing: Unifying segmentation, detection, and recognition," *International Journal of Computer Vision (IJCV)*, vol. 63, no. 2, pp. 113–140, 2005.

[153] D. Tuia, J. Muñoz-Marí, M. Kanevski, and G. Camps-Valls, "Structured output SVM for remote sensing image classification," *Journal of Signal Processing Systems*, 2010.

[154] V. Vapnik and A. Chervonenkis, *Theory of Pattern Recognition (in Russian)*. Nauka, Moscow, 1974.

[155] V. V. Vazirani, *Approximation Algorithms*. Springer, 2001.

[156] A. Vedaldi and A. Zisserman, "Structured output regression for detection with partial occulsion," in *Conference on Neural Information Processing Systems (NIPS)*, 2009.

[157] S. Vicente, V. Kolmogorov, and C. Rother, "Graph cut based image segmentation with connectivity priors," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[158] S. Vicente, V. Kolmogorov, and C. Rother, "Joint optimization of segmentation and appearance models," in *International Conference on Computer Vision (ICCV)*, 2009.

[159] S. V. N. Vishwanathan, N. N. Schraudolph, M. W. Schmidt, and K. P. Murphy, "Accelerated training of conditional random fields with stochastic gradient methods," in *International Conference on Machine Learing (ICML)*, pp. 969–976, 2006.

[160] M. J. Wainwright and M. I. Jordan, "Graphical models, exponential families and variational inference," *Foundations and Trends in Machine Learning*, vol. 1, no. 1-2, pp. 1–305, 2008.

[161] J. J. Weinman, L. Tran, and C. J. Pal, "Efficiently learning random fields for stereo vision with sparse message passing," in *European Conference on Computer Vision (ECCV)*, 2008.

[162] T. Werner, "A linear programming approach to max-sum problem: A review," *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, vol. 29, no. 7, pp. 1165–1179, 2007.

[163] T. Werner, "High-arity interactions, polyhedral relaxations, and cutting plane algorithm for soft constraint optimisation (MAP-MRF)," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[164] T. Werner, "Revisiting the decomposition approach to inference in exponential families and graphical models," Center for Machine Perception, Czech Technical University Prague, Research Report, CTU-CMP-2009-06, ftp://cmp.felk.cvut.cz/pub/cmp/articles/werner/Werner-TR-2009-06.pdf, 2009.

[165] T. Werner, "Belief propagation fixed points as zero gradients of a function of reparameterizations," Center for Machine Perception, Czech Technical University Prague, Research Report, CTU-CMP-2010-05, ftp://cmp.felk.cvut.cz/pub/cmp/articles/werner/Werner-TR-2010-05.pdf, 2010.

[166] H. P. Williams, *Model Building in Mathematical Programming.* New York: John Wiley & Sons, 4 Edition, 1999.

[167] J. Winn and C. M. Bishop, "Variational message passing," *Journal of Machine Learning Research (JMLR)*, vol. 6, pp. 661–694, 2005.

[168] L. A. Wolsey, *Integer Programming.* New York: John Wiley & Sons, 1998.

[169] O. J. Woodford, C. Rother, and V. Kolmogorov, "A global perspective on MAP inference for low-level vision," in *International Conference on Computer Vision (ICCV)*, 2009.

[170] E. P. Xing, M. I. Jordan, and S. J. Russell, "A generalized mean field algorithm for variational inference in exponential families," in *Uncertainty in Artificial Intelligence (UAI)*, pp. 583–591, 2003.

[171] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Constructing free energy approximations and generalized belief propagation algorithms," MERL Technical Report, 2004-040, http://www.merl.com/papers/docs/TR2004-040.pdf, 2004.

[172] C. N. J. Yu and T. Joachims, "Learning structural SVMs with latent variables," in *International Conference on Machine Learing (ICML)*, 2009.

[173] A. Yuille, "The convergence of contrastive divergences," in *Conference on Neural Information Processing Systems (NIPS)*, pp. 1593–1600, 2005.

[174] A. L. Yuille, "CCCP algorithms to minimize the Bethe and Kikuchi free energies: Convergent alternatives to belief propagation," *Neural Computation*, vol. 14, no. 7, pp. 1691–1722, 2002.

[175] A. L. Yuille and A. Rangarajan, "The concave-convex procedure," *Neural Computation*, vol. 15, no. 4, pp. 915–936, 2003.

[176] T. Zhang, "Statistical behavior and consistency of classification methods based on convex risk minimization," *Annals of Statistics*, vol. 32, no. 1, pp. 56–85, 2004.

[177] S. C. Zhu, Y. N. Wu, and D. Mumford, "Filters, random fields and maximum entropy (FRAME): Towards a unified theory for texture modeling," *International Journal of Computer Vision (IJCV)*, vol. 27, no. 2, pp. 107–126, 1998.