Chapter 4

Model fitting and optimization

R. Szeliski, *Computer Vision*, Texts in Computer Science,
https://doi.org/10.1007/978-3-030-34372-9_4

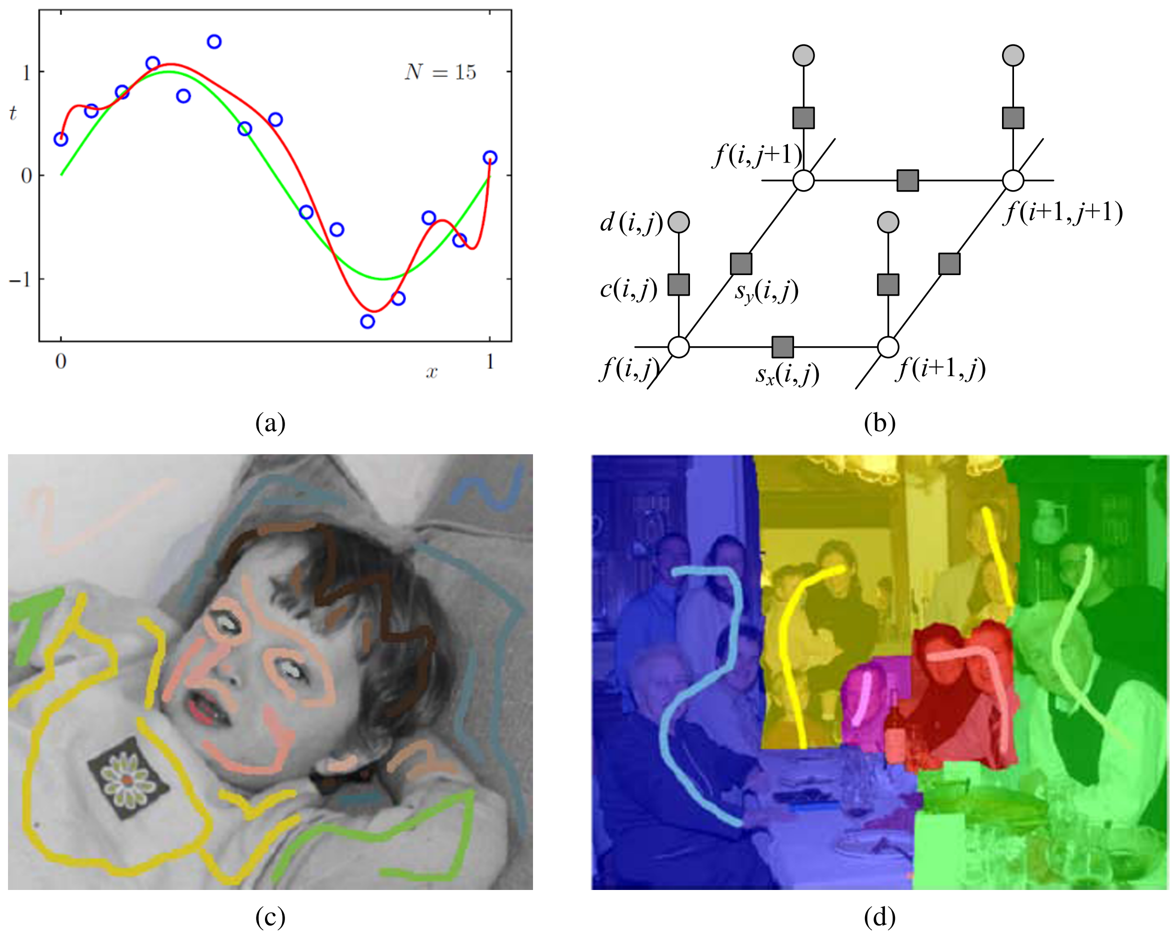


Figure 4.1 Examples of data interpolation and global optimization: (a) scattered data interpolation (curve fitting) (Bishop 2006) © 2006 Springer; (b) graphical model interpretation of first-order regularization; (c) colorization using optimization (Levin, Lischinski, and Weiss 2004) © 2004 ACM; (d) multi-image photomontage formulated as an unordered label MRF (Agarwala, Dontcheva *et al.* 2004) © 2004 ACM.

In the previous chapter, we covered a large number of image processing operators that take as input one or more images and produce some filtered or transformed version of these images. In many situations, however, we are given *incomplete* data as input, such as depths at a sparse number of locations, or user scribbles suggesting how an image should be colorized or segmented (Figure 4.1c–d).

The problem of interpolating a complete image (or more generally a *function* or *field*) from incomplete or varying quality data is often called *scattered data interpolation*. We begin this chapter with a review of techniques in this area, since in addition to being widely used in computer vision, they also form the basis of most machine learning algorithms, which we will study in the next chapter.

Instead of doing an exhaustive survey, we present in Section 4.1 some easy-to-use techniques, such as triangulation, spline interpolation, and radial basis functions. While these techniques are widely used, they cannot easily be modified to provide *controlled continuity*, i.e., to produce the kinds of piecewise continuous reconstructions we expect when estimating depth maps, label maps, or even color images.

For this reason, we introduce in Section 4.2 *variational methods*, which formulate the interpolation problem as the recovery of a piecewise smooth function subject to exact or approximate data constraints. Because the smoothness is controlled using penalties formulated as norms of the function, this class of techniques are often called *regularization* or *energy-based* approaches. To find the minimum-energy solutions to these problems, we discretize them (typically on a pixel grid), resulting in a discrete energy, which can then be minimized using sparse linear systems or related iterative techniques.

In the last part of this chapter, Section 4.3, we show how such energy-based formulations are related to Bayesian inference techniques formulated as *Markov random fields*, which are a special case of general probabilistic *graphical models*. In these formulations, data constraints can be interpreted as noisy and/or incomplete measurements, and piecewise smoothness constraints as *prior assumptions* or *models* over the solution space. Such formulations are also often called *generative models*, since we can, in principle, generate random samples from the prior distribution to see if they conform with our expectations. Because the prior models can be more complex than simple smoothness constraints, and because the solution space can have multiple local minima, more sophisticated optimization techniques have been developed, which we discuss in this section.

4.1 Scattered data interpolation

The goal of *scattered data interpolation* is to produce a (usually continuous and smooth) function $\mathbf{f}(\mathbf{x})$ that passes *through* a set of data points $\mathbf{d}_k$ placed at locations $\mathbf{x}_k$ such that

$$\mathbf{f}(\mathbf{x}_k) = \mathbf{d}_k. \tag{4.1}$$

The related problem of *scattered data approximation* only requires the function to pass *near* the data points (Amidror 2002; Wendland 2004; Anjyo, Lewis, and Pighin 2014). This is usually formulated using a penalty function such as

$$E_\text{D} = \sum_k \|\mathbf{f}(\mathbf{x}_k) - \mathbf{d}_k\|^2, \tag{4.2}$$

with the squared norm in the above formula sometimes replaced by a different norm or robust function (Section 4.1.3). In statistics and machine learning, the problem of predicting an output function

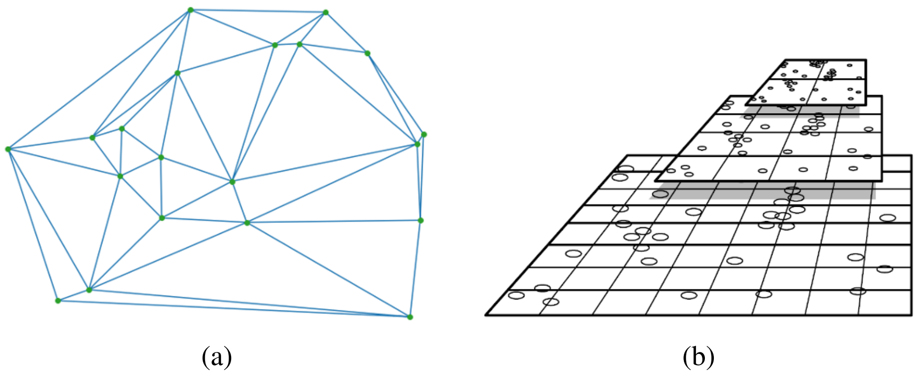


Figure 4.2 Some simple scattered data interpolation and approximation algorithms: (a) a Delaunay triangulation defined over a set of data point locations; (b) data structure and intermediate results for the *pull-push* algorithm (Gortler, Grzeszczuk *et al.* 1996) © 1996 ACM.

given a finite number of samples is called *regression* (Section 5.1). The $\mathbf{x}$ vectors are called the *inputs* and the outputs $\mathbf{y}$ are called the *targets*. Figure 4.1a shows an example of one-dimensional scattered data interpolation, while Figures 4.2 and 4.8 show some two-dimensional examples.

At first glance, scattered data interpolation seems closely related to *image interpolation*, which we studied in Section 3.5.1. However, unlike images, which are regularly gridded, the data points in scattered data interpolation are irregularly placed throughout the domain, as shown in Figure 4.2. This requires some adjustments to the interpolation methods we use.

If the domain $\mathbf{x}$ is two-dimensional, as is the case with images, one simple approach is to *triangulate* the domain $\mathbf{x}$ using the data locations $\mathbf{x}_k$ as the triangle vertices. The resulting triangular network, shown in Figure 4.2a, is called a *triangular irregular network* (TIN), and was one of the early techniques used to produce elevation maps from scattered field measurements collected by surveys.

The triangulation in Figure 4.2a was produced using a *Delaunay triangulation*, which is the most widely used planar triangulation technique due to its attractive computational properties, such as the avoidance of long skinny triangles. Algorithms for efficiently computing such triangulation are readily available[1] and covered in textbooks on computational geometry (Preparata and Shamos 1985; de Berg, Cheong *et al.* 2008). The Delaunay triangulation can be extended to higher-dimensional domains using the property of circumscribing spheres, i.e., the requirement that all selected simplices (triangles, tetrahedra, etc.) have no other vertices inside their circumscribing spheres.

Once the triangulation has been defined, it is straightforward to define a piecewise-linear interpolant over each triangle, resulting in an interpolant that is C_0 but not generally C_1 continuous. The formulas for the function inside each triangle are usually derived using *barycentric coordinates*, which attain their maximal values at the vertices and sum up to one (Farin 2002; Amidror 2002).

If a smoother surface is desired as the interpolant, we can replace the piecewise linear functions on each triangle with higher-order *splines*, much as we did for image interpolation (Section 3.5.1). However, since these splines are now defined over irregular triangulations, more sophisticated techniques must be used (Farin 2002; Amidror 2002). Other, more recent interpolators based on geometric modeling techniques in computer graphics include subdivision surfaces (Peters and Reif 2008).

An alternative to triangulating the data points is to use a regular n-dimensional grid, as shown in

[1]For example, https://docs.scipy.org/doc/scipy/reference/tutorial/spatial.html

Figure 4.2b. Splines defined on such domains are often called *tensor product splines* and have been used to interpolate scattered data (Lee, Wolberg, and Shin 1997).

An even faster, but less accurate, approach is called the *pull-push* algorithm and was originally developed for interpolating missing 4D lightfield samples in a Lumigraph (Gortler, Grzeszczuk *et al.* 1996). The algorithm proceeds in three phases, as schematically illustrated in Figure 4.2b.

First, the irregular data samples are *splatted* onto (i.e., spread across) the nearest grid vertices, using the same approach we discussed in Section 3.6.1 on parametric image transformations. The splatting operations accumulate both values and weights at nearby vertices. In the second, *pull*, phase, values and weights are computed at a hierarchical set of lower resolution grids by combining the coefficient values from the higher resolution grids. In the lower resolution grids, the gaps (regions where the weights are low) become smaller. In the third, *push*, phase, information from each lower resolution grid is combined with the next higher resolution grid, filling in the gaps while not unduly blurring the higher resolution information already computed. Details of these three stages can be found in (Gortler, Grzeszczuk *et al.* 1996).

The pull-push algorithm is very fast, since it is essentially linear in the number of input data points and fine-level grid samples.

4.1.1 Radial basis functions

While the mesh-based representations I have just described can provide good-quality interpolants, they are typically limited to low-dimensional domains, because the size of the mesh grows combinatorially with the dimensionality of the domain. In higher dimensions, it is common to use *mesh-free* approaches that define the desired interpolant as a weighted sum of basis functions, similar to the formulation used in image interpolation (3.64). In machine learning, such approaches are often called *kernel functions* or *kernel regression* (Bishop 2006, Chapter 6; Murphy 2012, Chapter 14; Schölkopf and Smola 2001).

In more detail, the interpolated function f is a weighted sum (or *superposition*) of basis functions centered at each input data point

$$\mathbf{f}(\mathbf{x}) = \sum_k \mathbf{w}_k \phi(\|\mathbf{x} - \mathbf{x}_k\|), \tag{4.3}$$

where the $\mathbf{x}_k$ are the locations of the scattered data points, the ϕs are the *radial basis functions* (or kernels), and $\mathbf{w}_k$ are the local *weights* associated with each kernel. The basis functions $\phi()$ are called *radial* because they are applied to the radial distance between a data sample $\mathbf{x}_k$ and an evaluation point $\mathbf{x}$. The choice of ϕ determines the smoothness properties of the interpolant, while the choice of weights w_k determines how closely the function approximates the input.

Some commonly used basis functions (Anjyo, Lewis, and Pighin 2014) include

Gaussian	$\phi(r) = \exp(-r^2/c^2)$	(4.4)
Hardy multiquadric	$\phi(r) = \sqrt{(}r^2 + c^2)$	(4.5)
Inverse multiquadric	$\phi(r) = 1/\sqrt{(}r^2 + c^2)$	(4.6)
Thin plate spline	$\phi(r) = r^2 \log r.$	(4.7)

In these equations, r is the radial distance and c is a scale parameter that controls the size (radial falloff) of the basis functions, and hence its smoothness (more compact bases lead to "peakier" solutions). The thin plate spline equation holds for two dimensions (the general n-dimensional spline is called the *polyharmonic spline* and is given in (Anjyo, Lewis, and Pighin 2014)) and is the analytic solution to the second degree variational spline derived in (4.19).

If we want our function to exactly interpolate the data values, we solve the linear system of equations (4.1), i.e.,

$$\mathbf{f}(\mathbf{x}_k) = \sum_l \mathbf{w}_l \phi(\|\mathbf{x}_k - \mathbf{x}_l\|) = \mathbf{d}_k, \tag{4.8}$$

to obtain the desired set of weights $\mathbf{w}_k$. Note that for large amounts of basis function overlap (large values of c), these equations may be quite *ill-conditioned*, i.e., small changes in data values or locations can result in large changes in the interpolated function. Note also that the solution of such a system of equations is in general $O(m^3)$, where m is the number of data points (unless we use basis functions with finite extent to obtain a sparse set of equations).

A more prudent approach is to solve the *regularized data approximation problem*, which involves minimizing the data constraint energy (4.2) together with a weight penalty (*regularizer*) of the form

$$E_{\mathrm{W}} = \sum_k \|\mathbf{w}_k\|^p, \tag{4.9}$$

and to then minimize the regularized least squares problem

$$E(\{w_k\}) = E_{\mathrm{D}} + \lambda E_{\mathrm{W}} \tag{4.10}$$

$$= \sum_k \|\sum_l \mathbf{w}_l \phi(\|\mathbf{x}_k - \mathbf{x}_l\|) - \mathbf{d}_k\|^2 + \lambda \sum_k \|\mathbf{w}_k\|^p. \tag{4.11}$$

When $p = 2$ (quadratic weight penalty), the resulting energy is a pure least squares problem, and can be solved using the *normal equations* (Appendix A.2), where the λ value gets added along the diagonal to stabilize the system of equations.

In statistics and machine learning, the quadratic (regularized least squares) problem is called *ridge regression*. In neural networks, adding a quadratic penalty on the weights is called *weight decay*, because it encourages weights to decay towards zero (Section 5.3.3). When $p = 1$, the technique is called *lasso* (least absolute shrinkage and selection operator), since for sufficiently large values of λ, many of the weights $\mathbf{w}_k$ get driven to zero (Tibshirani 1996; Bishop 2006; Murphy 2012; Deisenroth, Faisal, and Ong 2020). This results in a *sparse* set of basis functions being used in the interpolant, which can greatly speed up the computation of new values of $\mathbf{f}(\mathbf{x})$. We will have more to say on sparse kernel techniques in the section on Support Vector Machines (Section 5.1.4).

An alternative to solving a set of equations to determine the weights $\mathbf{w}_k$ is to simply set them to the input data values $\mathbf{d}_k$. However, this fails to interpolate the data, and instead produces higher values in higher density regions. This can be useful if we are trying to estimate a probability density function from a set of samples. In this case, the resulting density function, obtained after normalizing the sum of sample-weighted basis functions to have a unit integral, is called the *Parzen window* or *kernel* approach to probability density estimation (Duda, Hart, and Stork 2001, Section 4.3; Bishop 2006, Section 2.5.1). Such probability densities can be used, among other things, for (spatially) clustering color values together for image segmentation in what is known as the *mean shift* approach (Comaniciu and Meer 2002) (Section 7.5.2).

If, instead of just estimating a density, we wish to actually interpolate a set of data values $\mathbf{d}_k$, we can use a related technique known as *kernel regression* or the *Nadaraya-Watson* model, in which we divide the data-weighted summed basis functions by the sum of all the basis functions,

$$\mathbf{f}(\mathbf{x}) = \frac{\sum_k \mathbf{d}_k \phi(\|\mathbf{x} - \mathbf{x}_k\|)}{\sum_l \phi(\|\mathbf{x} - \mathbf{x}_l\|)}. \tag{4.12}$$

Note how this operation is similar, in concept, to the *splatting* method for forward rendering we discussed in Section 3.6.1, except that here, the bases can be much wider than the nearest-neighbor bilinear bases used in graphics (Takeda, Farsiu, and Milanfar 2007).

Kernel regression is equivalent to creating a new set of spatially varying normalized shifted basis functions

$$\phi'_k(\mathbf{x}) = \frac{\phi(\|\mathbf{x}-\mathbf{x}_k\|)}{\sum_l \phi(\|\mathbf{x}-\mathbf{x}_l\|)}, \tag{4.13}$$

which form a *partition of unity*, i.e., sum up to 1 at every location (Anjyo, Lewis, and Pighin 2014). While the resulting interpolant can now be written more succinctly as

$$\mathbf{f}(\mathbf{x}) = \sum_k \mathbf{d}_k \phi'_k(\|\mathbf{x}-\mathbf{x}_k\|), \tag{4.14}$$

in most cases, it is more expensive to precompute and store the K ϕ'_k functions than to evaluate (4.12).

While not that widely used in computer vision, kernel regression techniques have been applied by Takeda, Farsiu, and Milanfar (2007) to a number of low-level image processing operations, including state-of-the-art handheld multi-frame super-resolution (Wronski, Garcia-Dorado *et al.* 2019).

One last scattered data interpolation technique worth mentioning is *moving least squares*, where a weighted subset of nearby points is used to compute a local smooth surface. Such techniques are mostly widely used in 3D computer graphics, especially for point-based surface modeling, as discussed in Section 13.4 and (Alexa, Behr *et al.* 2003; Pauly, Keiser *et al.* 2003; Anjyo, Lewis, and Pighin 2014).

4.1.2 Overfitting and underfitting

When we introduced weight regularization in (4.9), we said that it was usually preferable to approximate the data but we did not explain why. In most data fitting problems, the samples $\mathbf{d}_k$ (and sometimes even their locations $\mathbf{x}_k$) are noisy, so that fitting them exactly makes no sense. In fact, doing so can introduce a lot of spurious wiggles, when the true solution is likely to be smoother.

To delve into this phenomenon, let us start with a simple polynomial fitting example taken from (Bishop 2006, Chapter 1.1). Figure 4.3 shows a number of polynomial curves of different orders M fit to the blue circles, which are noisy samples from the underlying green sine curve. Notice how the low-order ($M = 0$ and $M = 1$) polynomials severely *underfit* the underlying data, resulting in curves that are too flat, while the $M = 9$ polynomial, which exactly fits the data, exhibits far more wiggle than is likely.

How can we quantify this amount of underfitting and overfitting, and how can we get just the right amount? This topic is widely studied in machine learning and covered in a number of texts, including Bishop (2006, Chapter 1.1), Glassner (2018, Chapter 9), Deisenroth, Faisal, and Ong (2020, Chapter 8), and Zhang, Lipton *et al.* (2021, Section 4.4.3).

One approach is to use regularized least squares, introduced in (4.11). Figure 4.4 shows an $M = 9$th degree polynomial fit obtained by minimizing (4.11) with the polynomial basis functions $\phi_k(x) = x^k$ for two different values of λ. The left plot shows a reasonable amount of regularization, resulting in a plausible fit, while the larger value of λ on the right causes underfitting. Note that the $M = 9$ interpolant shown in the lower right quadrant of Figure 4.3 corresponds to the unregularized $\lambda = 0$ case.

If we were to now measure the difference between the red (estimated) and green (noise-free) curves, we see that choosing a good intermediate value of λ will produce the best result. In practice, however, we never have access to samples from the noise-free data.

Instead, if we are given a set of samples to interpolate, we can save some in a *validation* set in order to see if the function we compute is underfitting or overfitting. When we vary a parameter

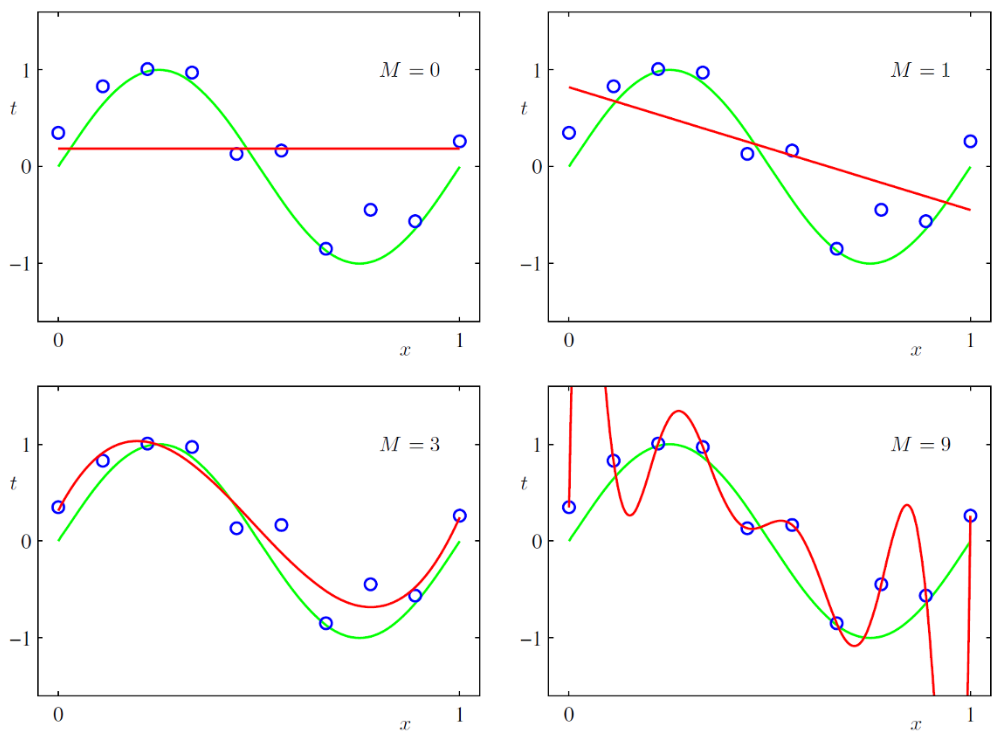


Figure 4.3 Polynomial curve fitting to the blue circles, which are noisy samples from the green sine curve (Bishop 2006) © 2006 Springer. The four plots show the 0th order constant function, the first order linear fit, the $M = 3$ cubic polynomial, and the 9th degree polynomial. Notice how the first two curves exhibit *underfitting*, while the last curve exhibits *overfitting*, i.e., excessive wiggle.

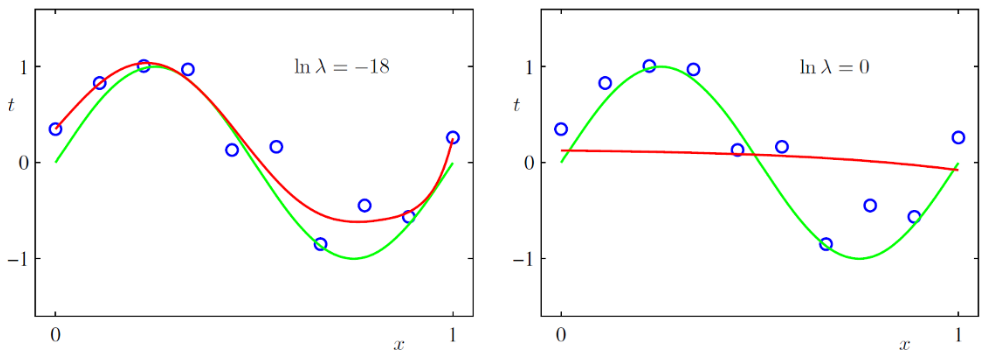


Figure 4.4 Regularized $M = 9$ polynomial fitting for two different values of λ (Bishop 2006) © 2006 Springer. The left plot shows a reasonable amount of regularization, resulting in a plausible fit, while the larger value of λ on the right causes underfitting.

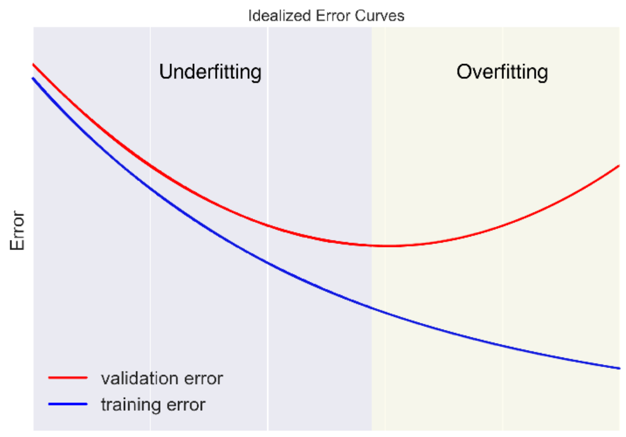


Figure 4.5 Fitting (training) and validation errors as a function of the amount of regularization or smoothing © Glassner (2018). The less regularized solutions on the right, while exhibiting lower fitting error, perform less well on the validation data.

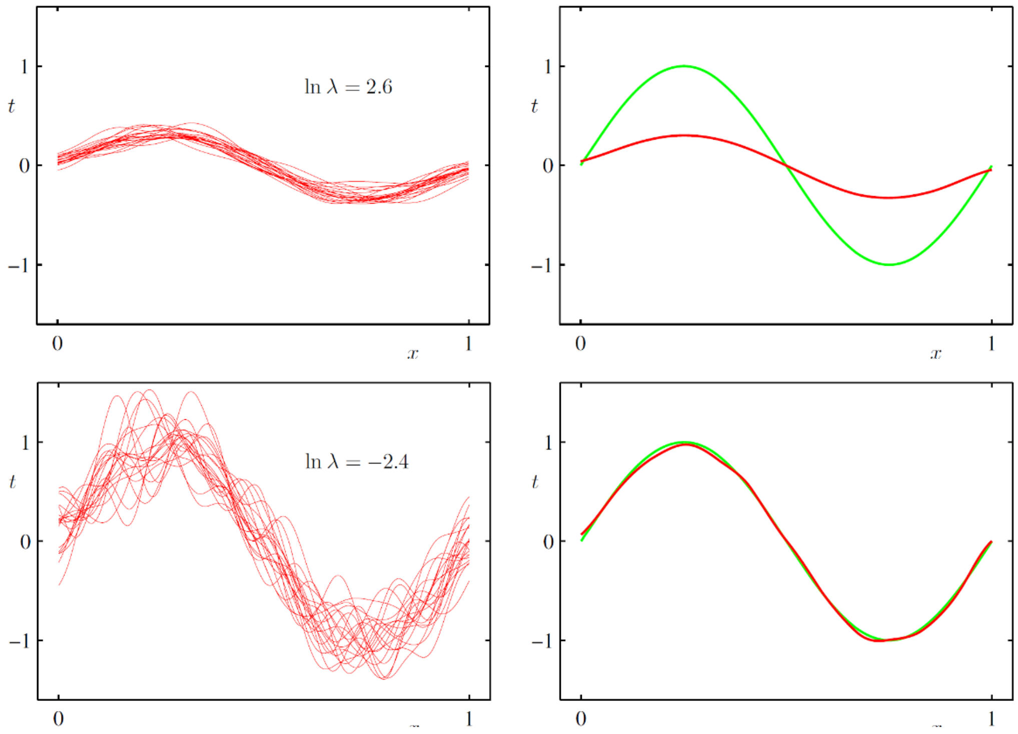


Figure 4.6 The more heavily regularized solution $\log \lambda = 2.6$ exhibits higher bias (deviation from original curve) than the less heavily regularized version ($\log \lambda = -2.4$), which has much higher variance (Bishop 2006) © 2006 Springer. The red curves on the left are $M = 24$ Gaussian basis fits to 25 randomly sampled points on the green curve. The red curve on the right is their mean.

such as λ (or use some other measure to control smoothness), we typically obtain a curve such as the one shown in Figure 4.5. In this figure, the blue curve denotes the fitting error, which in this case is called the *training error*, since in machine learning, we usually split the given data into a (typically larger) training set and a (typically smaller) validation set.

To obtain an even better estimate of the ideal amount of regularization, we can repeat the process of splitting our sample data into training and validation sets several times. One well-known technique, called *cross-validation* (Craven and Wahba 1979; Wahba and Wendelberger 1980; Bishop 2006, Section 1.3; Murphy 2012, Section 1.4.8; Deisenroth, Faisal, and Ong 2020, Chapter 8; Zhang, Lipton *et al.* 2021, Section 4.4.2), splits the training data into K *folds* (equal sized pieces). You then put aside each fold, in turn, and train on the remaining data. You can then estimate the best regularization parameter by averaging over all K training runs. While this generally works well ($K = 5$ is often used), it may be too expensive when training large neural networks because of the long training times involved.

Cross-validation is just one example of a class of *model selection* techniques that estimate *hyperparameters* in a training algorithm to achieve good performance. Additional methods include *information criteria* such as the Bayesian information criterion (BIC) (Torr 2002) and the Akaike information criterion (AIC) (Kanatani 1998), and Bayesian modeling approaches (Szeliski 1989; Bishop 2006; Murphy 2012).

One last topic worth mention with regard to data fitting, since it comes up often in discussions of statistical machine learning techniques, is the *bias-variance tradeoff* (Bishop 2006, Section 3.2). As you can see in Figure 4.6, using a large amount of regularization (top row) results in much lower variance between different random sample solutions, but much higher bias away from the true solution. Using insufficient regularization increases the variance dramatically, although an average over a large number of samples has low bias. The trick is to determine a reasonable compromise in terms of regularization so that any individual solution has a good expectation of being close to the *ground truth* (original clean continuous) data.

4.1.3 Robust data fitting

When we added a regularizer on the weights in (4.9), we noted that it did not have to be a quadratic penalty and could, instead, be a lower-order monomial that encouraged *sparsity* in the weights.

This same idea can be applied to data terms such as (4.2), where, instead of using a quadratic penalty, we can use a *robust loss function* $\rho()$,

$$E_{\rm R} = \sum_k \rho(\|\mathbf{r}_k\|), \quad \text{with } \mathbf{r}_k = \mathbf{f}(\mathbf{x}_k) - \mathbf{d}_k, \tag{4.15}$$

which gives lower weights to larger data fitting errors, which are more likely to be outlier measurements. (The fitting error term $\mathbf{r}_k$ is called the *residual error*.)

Some examples of loss functions from (Barron 2019) are shown in Figure 4.7 along with their derivatives. The regular quadratic ($\alpha = 2$) penalty gives full (linear) weight to each error, whereas the $\alpha = 1$ loss gives equal weight to all larger residuals, i.e., it behaves as an L_1 loss for large residuals, and L_2 for small ones. Even larger values of α discount large errors (outliers) even more, although they result in optimization problems that are *non-convex*, i.e., that can have multiple local minima. We will discuss techniques for finding good initial guesses for such problems later on in Section 8.1.4.

In statistics, minimizing non-quadratic loss functions to deal with potential outlier measurements is known as *M-estimation* (Huber 1981; Hampel, Ronchetti *et al.* 1986; Black and Rangarajan 1996; Stewart 1999). Such estimation problems are often solved using *iteratively reweighted least squares*,

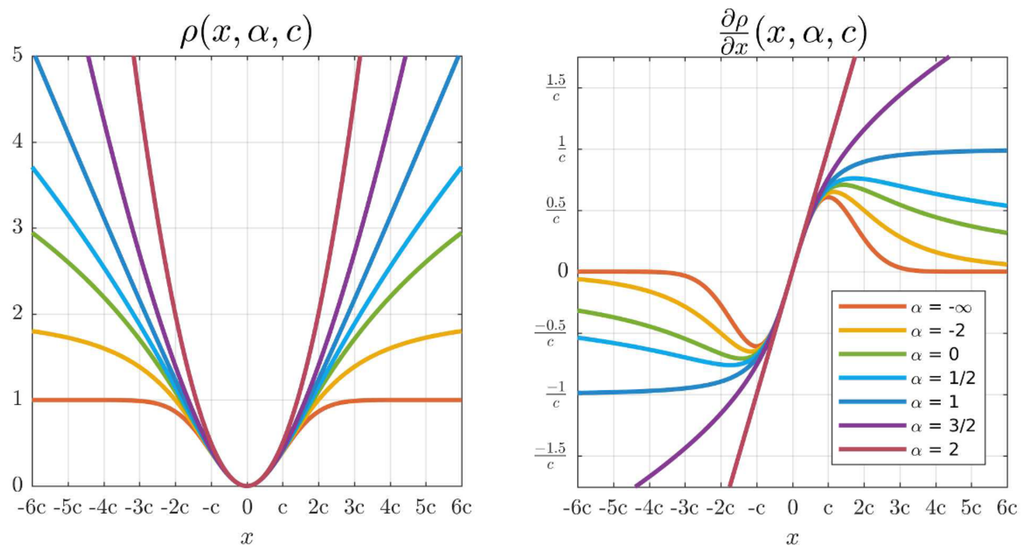


Figure 4.7 A general and adaptive loss function (left) and its gradient (right) for different values of its shape parameter α (Barron 2019) © 2019 IEEE. Several values of α reproduce existing loss functions: L_2 loss ($\alpha = 2$), Charbonnier loss ($\alpha = 1$), Cauchy loss ($\alpha = 0$), Geman-McClure loss ($\alpha = -2$), and Welsch loss ($\alpha = -1$).

which we discuss in more detail in Section 8.1.4 and Appendix B.3. The Appendix also discusses the relationship between robust statistics and non-Gaussian probabilistic models.

The generalized loss function introduced by Barron (2019) has two free parameters. The first one, α, controls how drastically outlier residuals are downweighted. The second (scale) parameter c controls the width of the quadratic well near the minimum, i.e., what range of residual values roughly corresponds to inliers. Traditionally, the choice of α, which corresponds to a variety of previously published loss functions, was determined heuristically, based on the expected shape of the outlier distribution and computational considerations (e.g., whether a convex loss was desired). The scale parameter c could be estimated using a robust measure of variance, as discussed in Appendix B.3.

In his paper, Barron (2019) discusses how both parameters can be determined at run time by maximizing the likelihood (or equivalently, minimizing the negative log-likelihood) of the given residuals, making such an algorithm self-tuning to a wide variety of noise levels and outlier distributions.

4.2 Variational methods and regularization

The theory of regularization we introduced in the previous section was first developed by statisticians trying to fit models to data that severely underconstrained the solution space (Tikhonov and Arsenin 1977; Engl, Hanke, and Neubauer 1996). Consider, for example, finding a smooth surface that passes through (or near) a set of measured data points (Figure 4.8). Such a problem is described as *ill-posed* because many possible surfaces can fit this data. Since small changes in the input can sometimes lead to large changes in the fit (e.g., if we use polynomial interpolation), such problems are also often *ill-conditioned*. Since we are trying to recover the unknown function $f(x, y)$ from which the data points $d(x_i, y_i)$ were sampled, such problems are also often called *inverse problems*. Many computer vision tasks can be viewed as inverse problems, since we are trying to recover a full description of the 3D world from a limited set of images.

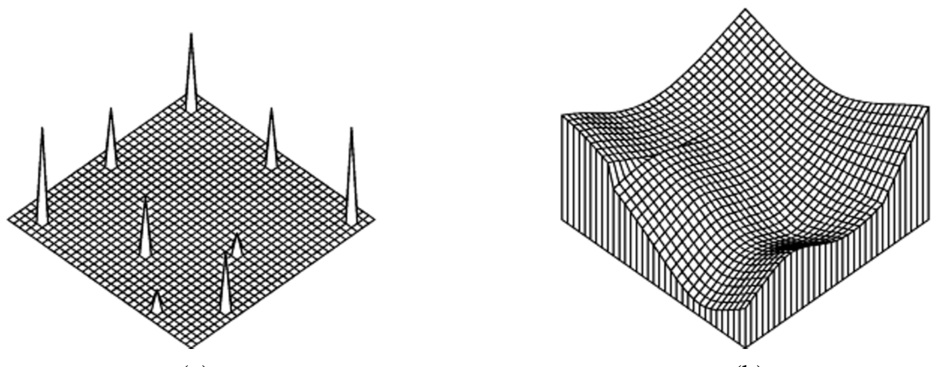


Figure 4.8 A simple surface interpolation problem: (a) nine data points of various heights scattered on a grid; (b) second-order, controlled-continuity, thin-plate spline interpolator, with a tear along its left edge and a crease along its right (Szeliski 1989) © 1989 Springer.

In the previous section, we attacked this problem using basis functions placed at the data points, or other heuristics such as the pull-push algorithm. While such techniques can provide reasonable solutions, they do not let us directly *quantify* and hence *optimize* the amount of smoothness in the solution, nor do they give us local control over where the solution should be discontinuous (Figure 4.8).

To do this, we use norms (measures) on function derivatives (described below) to formulate the problem and then find minimal energy solutions to these norms. Such techniques are often called *energy-based* or *optimization-based* approaches to computer vision. They are also often called *variational*, since we can use the *calculus of variations* to find the optimal solutions. Variational methods have been widely used in computer vision since the early 1980s to pose and solve a number of fundamental problems, including optical flow (Horn and Schunck 1981; Black and Anandan 1993; Brox, Bruhn *et al.* 2004; Werlberger, Pock, and Bischof 2010), segmentation (Kass, Witkin, and Terzopoulos 1988; Mumford and Shah 1989; Chan and Vese 2001), denoising (Rudin, Osher, and Fatemi 1992; Chan, Osher, and Shen 2001; Chan and Shen 2005), and multi-view stereo (Faugeras and Keriven 1998; Pons, Keriven, and Faugeras 2007; Kolev, Klodt *et al.* 2009). A more detailed list of relevant papers can be found in the Additional Reading section at the end of this chapter.

In order to quantify what it means to find a smooth solution, we can define a norm on the solution space. For one-dimensional functions $f(x)$, we can integrate the squared first derivative of the function,

$$\mathcal{E}_1 = \int f_x^2(x)\, dx \tag{4.16}$$

or perhaps integrate the squared second derivative,

$$\mathcal{E}_2 = \int f_{xx}^2(x)\, dx. \tag{4.17}$$

(Here, we use subscripts to denote differentiation.) Such energy measures are examples of *functionals*, which are operators that map functions to scalar values. They are also often called *variational methods*, because they measure the variation (non-smoothness) in a function.

In two dimensions (e.g., for images, flow fields, or surfaces), the corresponding smoothness functionals are

$$\mathcal{E}_1 = \int f_x^2(x,y) + f_y^2(x,y)\, dx\, dy = \int \|\nabla f(x,y)\|^2\, dx\, dy \tag{4.18}$$

and

$$\mathcal{E}_2 = \int f_{xx}^2(x,y) + 2f_{xy}^2(x,y) + f_{yy}^2(x,y)\,dx\,dy, \tag{4.19}$$

where the mixed $2f_{xy}^2$ term is needed to make the measure rotationally invariant (Grimson 1983).

The first derivative norm is often called the *membrane*, since interpolating a set of data points using this measure results in a tent-like structure. (In fact, this formula is a small-deflection approximation to the surface area, which is what soap bubbles minimize.) The second-order norm is called the *thin-plate spline*, since it approximates the behavior of thin plates (e.g., flexible steel) under small deformations. A blend of the two is called the *thin-plate spline under tension* (Terzopoulos 1986b).

The regularizers (smoothness functions) we have just described force the solution to be smooth and C_0 and/or C_1 continuous everywhere. In most computer vision applications, however, the fields we are trying to model or recover are only piecewise continuous, e.g., depth maps and optical flow fields jump at object discontinuities. Color images are even more discontinuous, since they also change appearance at albedo (surface color) and shading discontinuities.

To better model such functions, Terzopoulos (1986b) introduced *controlled-continuity splines*, where each derivative term is multiplied by a local weighting function,

$$\begin{aligned}\mathcal{E}_{\mathrm{CC}} = \int \rho(x,y)\{&[1-\tau(x,y)][f_x^2(x,y) + f_y^2(x,y)] \\ &+ \tau(x,y)[f_{xx}^2(x,y) + 2f_{xy}^2(x,y) + f_{yy}^2(x,y)]\}\,dx\,dy.\end{aligned} \tag{4.20}$$

Here, $\rho(x,y) \in [0,1]$ controls the *continuity* of the surface and $\tau(x,y) \in [0,1]$ controls the local *tension*, i.e., how flat the surface wants to be. Figure 4.8 shows a simple example of a controlled-continuity interpolator fit to nine scattered data points. In practice, it is more common to find first-order smoothness terms used with images and flow fields (Section 9.3) and second-order smoothness associated with surfaces (Section 13.3.1).

In addition to the smoothness term, variational problems also require a data term (or *data penalty*). For scattered data interpolation (Nielson 1993), the data term measures the distance between the function $f(x,y)$ and a set of data points $d_i = d(x_i, y_i)$,

$$\mathcal{E}_{\mathrm{D}} = \sum_i [f(x_i, y_i) - d_i]^2. \tag{4.21}$$

For a problem like noise removal, a continuous version of this measure can be used,

$$\mathcal{E}_{\mathrm{D}} = \int [f(x,y) - d(x,y)]^2\,dx\,dy. \tag{4.22}$$

To obtain a global energy that can be minimized, the two energy terms are usually added together,

$$\mathcal{E} = \mathcal{E}_{\mathrm{D}} + \lambda\mathcal{E}_{\mathrm{S}}, \tag{4.23}$$

where $\mathcal{E}_{\mathrm{S}}$ is the *smoothness penalty* ($\mathcal{E}_1$, $\mathcal{E}_2$ or some weighted blend such as $\mathcal{E}_{\mathrm{CC}}$) and λ is the *regularization parameter*, which controls the smoothness of the solution. As we saw in Section 4.1.2, good values for the regularization parameter can be estimated using techniques such as cross-validation.

4.2.1 Discrete energy minimization

In order to find the minimum of this continuous problem, the function $f(x, y)$ is usually first discretized on a regular grid.[2] The most principled way to perform this discretization is to use *finite element analysis*, i.e., to approximate the function with a piecewise continuous spline, and then perform the analytic integration (Bathe 2007).

Fortunately, for both the first-order and second-order smoothness functionals, the judicious selection of appropriate finite elements results in particularly simple discrete forms (Terzopoulos 1983). The corresponding *discrete* smoothness energy functions become

$$\begin{aligned} E_1 = \sum_{i,j} & s_x(i,j)[f(i+1,j) - f(i,j) - g_x(i,j)]^2 \\ & + s_y(i,j)[f(i,j+1) - f(i,j) - g_y(i,j)]^2 \end{aligned} \tag{4.24}$$

and

$$\begin{aligned} E_2 = h^{-2} \sum_{i,j} & c_x(i,j)[f(i+1,j) - 2f(i,j) + f(i-1,j)]^2 \\ & + 2c_m(i,j)[f(i+1,j+1) - f(i+1,j) - f(i,j+1) + f(i,j)]^2 \\ & + c_y(i,j)[f(i,j+1) - 2f(i,j) + f(i,j-1)]^2, \end{aligned} \tag{4.25}$$

where h is the size of the finite element grid. The h factor is only important if the energy is being discretized at a variety of resolutions, as in coarse-to-fine or multigrid techniques.

The optional smoothness weights $s_x(i,j)$ and $s_y(i,j)$ control the location of horizontal and vertical tears (or weaknesses) in the surface. For other problems, such as colorization (Levin, Lischinski, and Weiss 2004) and interactive tone mapping (Lischinski, Farbman *et al.* 2006), they control the smoothness in the interpolated chroma or exposure field and are often set inversely proportional to the local luminance gradient strength. For second-order problems, the crease variables $c_x(i,j)$, $c_m(i,j)$, and $c_y(i,j)$ control the locations of creases in the surface (Terzopoulos 1988; Szeliski 1990a).

The data values $g_x(i,j)$ and $g_y(i,j)$ are gradient data terms (constraints) used by algorithms, such as photometric stereo (Section 13.1.1), HDR tone mapping (Section 10.2.1) (Fattal, Lischinski, and Werman 2002), Poisson blending (Section 8.4.4) (Pérez, Gangnet, and Blake 2003), gradient-domain blending (Section 8.4.4) (Levin, Zomet *et al.* 2004), and Poisson surface reconstruction (Section 13.5.1) (Kazhdan, Bolitho, and Hoppe 2006; Kazhdan and Hoppe 2013). They are set to zero when just discretizing the conventional first-order smoothness functional (4.18). Note how separate smoothness and curvature terms can be imposed in the x, y, and mixed directions to produce local tears or creases (Terzopoulos 1988; Szeliski 1990a).

The two-dimensional discrete data energy is written as

$$E_\text{D} = \sum_{i,j} c(i,j)[f(i,j) - d(i,j)]^2, \tag{4.26}$$

where the local confidence weights $c(i,j)$ control how strongly the data constraint is enforced. These values are set to zero where there is no data and can be set to the inverse variance of the data measurements when there is data (as discussed by Szeliski (1989) and in Section 4.3).

The total energy of the discretized problem can now be written as a *quadratic form*

$$E = E_\text{D} + \lambda E_\text{S} = \mathbf{x}^T \mathbf{A} \mathbf{x} - 2\mathbf{x}^T \mathbf{b} + c, \tag{4.27}$$

[2]The alternative of using *kernel basis functions* centered on the data points (Boult and Kender 1986; Nielson 1993) is discussed in more detail in Section 13.3.1.

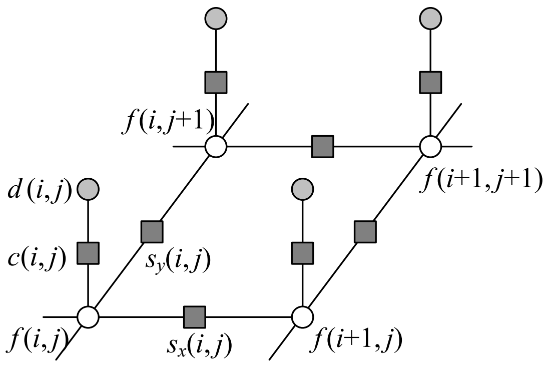


Figure 4.9 Graphical model interpretation of first-order regularization. The white circles are the unknowns $f(i,j)$ while the dark circles are the input data $d(i,j)$. In the resistive grid interpretation, the d and f values encode input and output voltages and the black squares denote resistors whose *conductance* is set to $s_x(i,j)$, $s_y(i,j)$, and $c(i,j)$. In the spring-mass system analogy, the circles denote elevations and the black squares denote springs. The same graphical model can be used to depict a first-order Markov random field (Figure 4.12).

where $\mathbf{x} = [f(0,0) \ldots f(m-1,n-1)]$ is called the *state vector*.[3]

The sparse symmetric positive-definite matrix $\mathbf{A}$ is called the *Hessian* since it encodes the second derivative of the energy function.[4] For the one-dimensional, first-order problem, $\mathbf{A}$ is tridiagonal; for the two-dimensional, first-order problem, it is multi-banded with five non-zero entries per row. We call $\mathbf{b}$ the *weighted data vector*. Minimizing the above quadratic form is equivalent to solving the sparse linear system

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{4.28}$$

which can be done using a variety of sparse matrix techniques, such as multigrid (Briggs, Henson, and McCormick 2000) and hierarchical preconditioners (Szeliski 2006b; Krishnan and Szeliski 2011; Krishnan, Fattal, and Szeliski 2013), as described in Appendix A.5 and illustrated in Figure 4.11. Using such techniques is essential to obtaining reasonable run-times, since properly preconditioned sparse linear systems have convergence times that are *linear* in the number of pixels.

While regularization was first introduced to the vision community by Poggio, Torre, and Koch (1985) and Terzopoulos (1986b) for problems such as surface interpolation, it was quickly adopted by other vision researchers for such varied problems as edge detection (Section 7.2), optical flow (Section 9.3), and shape from shading (Section 13.1) (Poggio, Torre, and Koch 1985; Horn and Brooks 1986; Terzopoulos 1986b; Bertero, Poggio, and Torre 1988; Brox, Bruhn *et al.* 2004). Poggio, Torre, and Koch (1985) also showed how the discrete energy defined by Equations (4.24–4.26) could be implemented in a resistive grid, as shown in Figure 4.9. In computational photography (Chapter 10), regularization and its variants are commonly used to solve problems such as high-dynamic range tone mapping (Fattal, Lischinski, and Werman 2002; Lischinski, Farbman *et al.* 2006), Poisson and gradient-domain blending (Pérez, Gangnet, and Blake 2003; Levin, Zomet *et al.* 2004; Agarwala, Dontcheva *et al.* 2004), colorization (Levin, Lischinski, and Weiss 2004), and natural image matting (Levin, Lischinski, and Weiss 2008).

[3]We use $\mathbf{x}$ instead of $\mathbf{f}$ because this is the more common form in the numerical analysis literature (Golub and Van Loan 1996).

[4]In numerical analysis, $\mathbf{A}$ is called the *coefficient* matrix (Saad 2003); in finite element analysis (Bathe 2007), it is called the *stiffness* matrix.

Robust regularization

While regularization is most commonly formulated using quadratic (L_2) norms, i.e., the squared derivatives in (4.16–4.19) and squared differences in (4.24–4.25), it can also be formulated using the non-quadratic *robust* penalty functions first introduced in Section 4.1.3 and discussed in more detail in Appendix B.3. For example, (4.24) can be generalized to

$$\begin{aligned} E_{1\mathrm{R}} = \sum_{i,j} & s_x(i,j)\rho(f(i+1,j) - f(i,j)) \\ & + s_y(i,j)\rho(f(i,j+1) - f(i,j)), \end{aligned} \tag{4.29}$$

where $\rho(x)$ is some monotonically increasing penalty function. For example, the family of norms $\rho(x) = |x|^p$ is called p-norms. When $p < 2$, the resulting smoothness terms become more piecewise continuous than totally smooth, which can better model the discontinuous nature of images, flow fields, and 3D surfaces.

An early example of robust regularization is the *graduated non-convexity* (GNC) algorithm of Blake and Zisserman (1987). Here, the norms on the data and derivatives are clamped,

$$\rho(x) = \min(x^2, V). \tag{4.30}$$

Because the resulting problem is highly non-convex (it has many local minima), a *continuation* method is proposed, where a quadratic norm (which is convex) is gradually replaced by the non-convex robust norm (Allgower and Georg 2003). (Around the same time, Terzopoulos (1988) was also using continuation to infer the tear and crease variables in his surface interpolation problems.)

4.2.2 Total variation

Today, many regularized problems are formulated using the L_1 ($p = 1$) norm, which is often called *total variation* (Rudin, Osher, and Fatemi 1992; Chan, Osher, and Shen 2001; Chambolle 2004; Chan and Shen 2005; Tschumperlé and Deriche 2005; Tschumperlé 2006; Cremers 2007; Kaftory, Schechner, and Zeevi 2007; Kolev, Klodt *et al.* 2009; Werlberger, Pock, and Bischof 2010). The advantage of this norm is that it tends to better preserve discontinuities, but still results in a convex problem that has a globally unique solution. Other norms, for which the *influence* (derivative) more quickly decays to zero, are presented by Black and Rangarajan (1996), Black, Sapiro *et al.* (1998), and Barron (2019) and discussed in Section 4.1.3 and Appendix B.3.

Even more recently, *hyper-Laplacian* norms with $p < 1$ have gained popularity, based on the observation that the log-likelihood distribution of image derivatives follows a $p \approx 0.5 - 0.8$ slope and is therefore a hyper-Laplacian distribution (Simoncelli 1999; Levin and Weiss 2007; Weiss and Freeman 2007; Krishnan and Fergus 2009). Such norms have an even stronger tendency to prefer large discontinuities over small ones. See the related discussion in Section 4.3 (4.43).

While least squares regularized problems using L_2 norms can be solved using linear systems, other p-norms require different iterative techniques, such as iteratively reweighted least squares (IRLS), Levenberg–Marquardt, alternation between local non-linear subproblems and global quadratic regularization (Krishnan and Fergus 2009), or primal-dual algorithms (Chambolle and Pock 2011). Such techniques are discussed in Section 8.1.3 and Appendices A.3 and B.3.

4.2.3 Bilateral solver

In our discussion of variational methods, we have focused on energy minimization problems based on gradients and higher-order derivatives, which in the discrete setting involves evaluating weighted

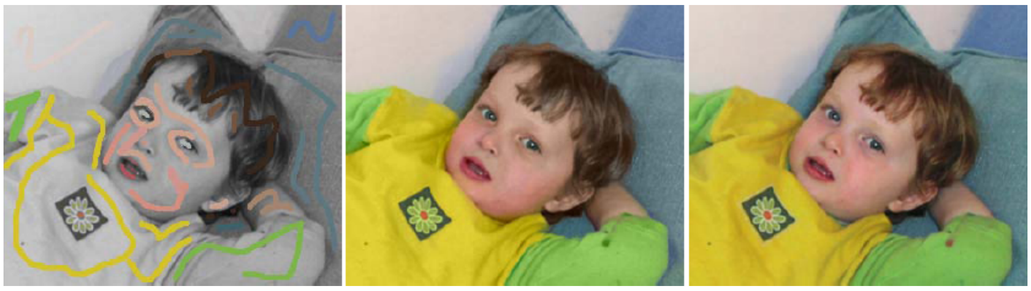


Figure 4.10 Colorization using optimization (Levin, Lischinski, and Weiss 2004) © 2004 ACM: (a) grayscale image with some color scribbles overlaid; (b) resulting colorized image; (c) original color image from which the grayscale image and the chrominance values for the scribbles were derived. Original photograph by Rotem Weiss.

errors between neighboring pixels. As we saw previously in our discussion of bilateral filtering in Section 3.3.2, we can often get better results by looking at a larger spatial neighborhood and combining pixels with similar colors or grayscale values. To extend this idea to a variational (energy minimization) setting, Barron and Poole (2016) propose replacing the usual first-order nearest-neighbor smoothness penalty (4.24) with a wider-neighborhood, bilaterally weighted version

$$E_{\mathrm{B}} = \sum_{i,j} \sum_{k,l} \hat{w}(i,j,k,l)[f(k,l) - f(i,j)]^2, \tag{4.31}$$

where

$$\hat{w}(i,j,k,l) = \frac{w(i,j,k,l)}{\sum_{m,n} w(i,j,m,n)}, \tag{4.32}$$

is the *bistochastized* (normalized) version of the *bilateral weight function* given in (3.37), which may depend on an input guide image, but not on the estimated values of f.[5]

To efficiently solve the resulting set of equations (which are much denser than nearest-neighbor versions), the authors use the same approach originally used to accelerate bilateral filtering, i.e., solving a related problem on a (spatially coarser) bilateral grid. The sequence of operations resembles those used for bilateral filtering, except that after splatting and before slicing, an iterative least squares solver is used instead of a multi-dimensional Gaussian blur. To further speed up the conjugate gradient solver, Barron and Poole (2016) use a multi-level preconditioner inspired by previous work on image-adapted preconditioners (Szeliski 2006b; Krishnan, Fattal, and Szeliski 2013).

Since its introduction, the bilateral solver has been used in a number of video processing and 3D reconstruction applications, including the stitching of binocular omnidirectional panoramic videos (Anderson, Gallup *et al.* 2016). The smartphone AR system developed by Valentin, Kowdle *et al.* (2018) extends the bilateral solver to have local *planar* models and uses a hardware-friendly real-time implementation (Mazumdar, Alaghi *et al.* 2017) to produce dense occlusion effects.

4.2.4 *Application*: Interactive colorization

A good use of edge-aware interpolation techniques is in *colorization*, i.e., manually adding colors to a "black and white" (grayscale) image. In most applications of colorization, the user draws some scribbles indicating the desired colors in certain regions (Figure 4.10a) and the system interpolates

[5]Note that in their paper, Barron and Poole (2016) use different σ_r values for the luminance and chrominance components of pixel color differences.

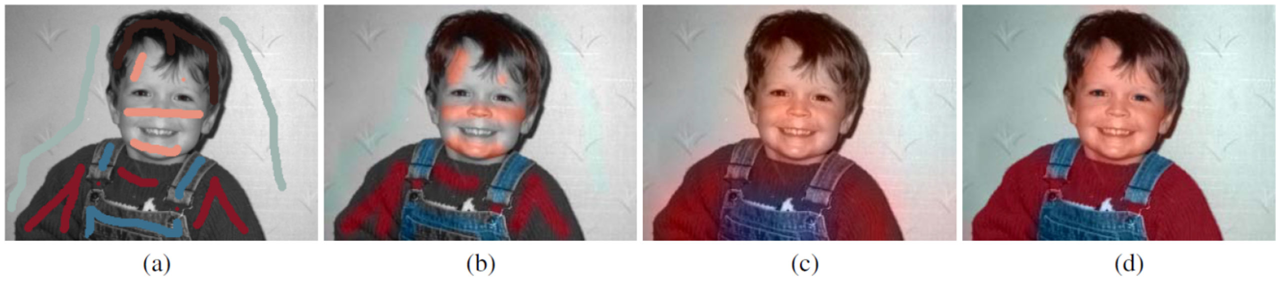


Figure 4.11 Speeding up the inhomogeneous least squares colorization solver using locally adapted hierarchical basis preconditioning (Szeliski 2006b) © 2006 ACM: (a) input gray image with color strokes overlaid; (b) solution after 20 iterations of conjugate gradient; (c) using one iteration of hierarchical basis function preconditioning; (d) using one iteration of locally adapted hierarchical basis functions.

the specified chrominance (u, v) values to the whole image, which are then re-combined with the input luminance channel to produce a final colorized image, as shown in Figure 4.10b. In the system developed by Levin, Lischinski, and Weiss (2004), the interpolation is performed using locally weighted regularization (4.24), where the local smoothness weights are inversely proportional to luminance gradients. This approach to locally weighted regularization has inspired later algorithms for high dynamic range tone mapping (Lischinski, Farbman *et al.* 2006)(Section 10.2.1, as well as other applications of the weighted least squares (WLS) formulation (Farbman, Fattal *et al.* 2008). These techniques have benefitted greatly from image-adapted regularization techniques, such as those developed in Szeliski (2006b), Krishnan and Szeliski (2011), Krishnan, Fattal, and Szeliski (2013), and Barron and Poole (2016), as shown in Figure 4.11. An alternative approach to performing the sparse chrominance interpolation based on geodesic (edge-aware) distance functions has been developed by Yatziv and Sapiro (2006). Neural networks can also be used to implement *deep priors* for image colorization (Zhang, Zhu *et al.* 2017).

4.3 Markov random fields

As we have just seen, regularization, which involves the minimization of energy functionals defined over (piecewise) continuous functions, can be used to formulate and solve a variety of low-level computer vision problems. An alternative technique is to formulate a *Bayesian* or *generative* model, which separately models the noisy image formation (*measurement*) process, as well as assuming a statistical *prior* model over the solution space (Bishop 2006, Section 1.5.4). In this section, we look at priors based on Markov random fields, whose log-likelihood can be described using local neighborhood interaction (or penalty) terms (Kindermann and Snell 1980; Geman and Geman 1984; Marroquin, Mitter, and Poggio 1987; Li 1995; Szeliski, Zabih *et al.* 2008; Blake, Kohli, and Rother 2011).

The use of Bayesian modeling has several potential advantages over regularization (see also Appendix B). The ability to model measurement processes statistically enables us to extract the maximum information possible from each measurement, rather than just guessing what weighting to give the data. Similarly, the parameters of the prior distribution can often be *learned* by observing samples from the class we are modeling (Roth and Black 2007a; Tappen 2007; Li and Huttenlocher 2008). Furthermore, because our model is probabilistic, it is possible to estimate (in principle) complete probability *distributions* over the unknowns being recovered and, in particular, to model the *uncertainty* in the solution, which can be useful in later processing stages. Finally, Markov random field models can be defined over *discrete* variables, such as image labels (where the variables have no proper ordering), for which regularization does not apply.

According to Bayes' rule (Appendix B.4), the *posterior* distribution $p(\mathbf{x}|\mathbf{y})$ over the unknowns $\mathbf{x}$ given the measurements $\mathbf{y}$ can be obtained by multiplying the measurement likelihood $p(\mathbf{y}|\mathbf{x})$ by the prior distribution $p(\mathbf{x})$ and normalizing,

$$p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{x})p(\mathbf{x})}{p(\mathbf{y})}, \tag{4.33}$$

where $p(\mathbf{y}) = \int_{\mathbf{x}} p(\mathbf{y}|\mathbf{x})p(\mathbf{x})$ is a normalizing constant used to make the $p(\mathbf{x}|\mathbf{y})$ distribution *proper* (integrate to 1). Taking the negative logarithm of both sides of (4.33), we get

$$-\log p(\mathbf{x}|\mathbf{y}) = -\log p(\mathbf{y}|\mathbf{x}) - \log p(\mathbf{x}) + C, \tag{4.34}$$

which is the *negative posterior log likelihood*.

To find the most likely (*maximum a posteriori* or MAP) solution $\mathbf{x}$ given some measurements $\mathbf{y}$, we simply minimize this negative log likelihood, which can also be thought of as an *energy*,

$$E(\mathbf{x}, \mathbf{y}) = E_{\mathrm{D}}(\mathbf{x}, \mathbf{y}) + E_{\mathrm{P}}(\mathbf{x}). \tag{4.35}$$

(We drop the constant C because its value does not matter during energy minimization.) The first term $E_{\mathrm{D}}(\mathbf{x}, \mathbf{y})$ is the *data energy* or *data penalty*; it measures the negative log likelihood that the data were observed given the unknown state $\mathbf{x}$. The second term $E_{\mathrm{P}}(\mathbf{x})$ is the *prior energy*; it plays a role analogous to the smoothness energy in regularization. Note that the MAP estimate may not always be desirable, as it selects the "peak" in the posterior distribution rather than some more stable statistic—see the discussion in Appendix B.2 and by Levin, Weiss *et al.* (2009).

For the remainder of this section, we focus on Markov random fields, which are probabilistic models defined over two or three-dimensional pixel or voxel grids. Before we dive into this, however, we should mention that MRFs are just one special case of the more general family of *graphical models* (Bishop 2006, Chapter 8; Koller and Friedman 2009; Nowozin and Lampert 2011; Murphy 2012, Chapters 10, 17, 19), which have sparse interactions between variables that can be captured in a *factor graph* (Dellaert and Kaess 2017; Dellaert 2021), such as the one shown in Figure 4.12. Graphical models come in a wide variety of topologies, including chains (used for audio and speech processing), trees (often used for modeling kinematic chains in tracking people (e.g., Felzenszwalb and Huttenlocher 2005)), stars (simplified models for people; Dalal and Triggs 2005; Felzenszwalb, Girshick *et al.* 2010, and constellations (Fergus, Perona, and Zisserman 2007). Such models were widely used for part-based recognition, as discussed in Section 6.2.1. For graphs that are acyclic, efficient linear-time inference algorithms based on dynamic programming can be used.

For image processing applications, the unknowns $\mathbf{x}$ are the set of output pixels

$$\mathbf{x} = [f(0,0) \ldots f(m-1, n-1)], \tag{4.36}$$

and the data are (in the simplest case) the input pixels

$$\mathbf{y} = [d(0,0) \ldots d(m-1, n-1)] \tag{4.37}$$

as shown in Figure 4.12.

For a Markov random field, the probability $p(\mathbf{x})$ is a *Gibbs* or *Boltzmann distribution*, whose negative log likelihood (according to the Hammersley–Clifford theorem) can be written as a sum of pairwise interaction potentials,

$$E_{\mathrm{P}}(\mathbf{x}) = \sum_{\{(i,j),(k,l)\} \in \mathcal{N}(i,j)} V_{i,j,k,l}(f(i,j), f(k,l)), \tag{4.38}$$

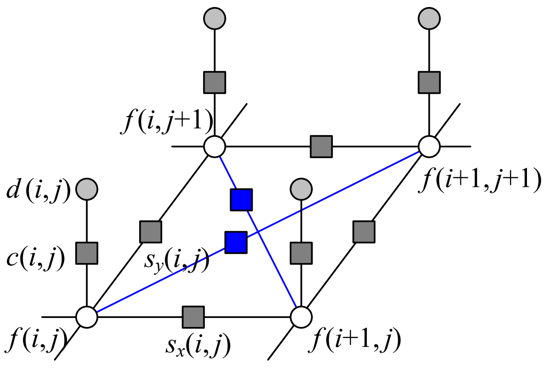


Figure 4.12 Graphical model for an $\mathcal{N}_4$ neighborhood Markov random field. (The blue edges are added for an $\mathcal{N}_8$ neighborhood.) The white circles are the unknowns $f(i, j)$, while the dark circles are the input data $d(i, j)$. The $s_x(i, j)$ and $s_y(i, j)$ black boxes denote arbitrary *interaction potentials* between adjacent nodes in the random field, and the $c(i, j)$ denote the *data penalty* functions. The same graphical model can be used to depict a discrete version of a first-order regularization problem (Figure 4.9).

where $\mathcal{N}(i, j)$ denotes the *neighbors* of pixel (i, j). In fact, the general version of the theorem says that the energy may have to be evaluated over a larger set of *cliques*, which depend on the *order* of the Markov random field (Kindermann and Snell 1980; Geman and Geman 1984; Bishop 2006; Kohli, Ladický, and Torr 2009; Kohli, Kumar, and Torr 2009).

The most commonly used neighborhood in Markov random field modeling is the $\mathcal{N}_4$ neighborhood, where each pixel in the field $f(i, j)$ interacts only with its immediate neighbors. The model in Figure 4.12, which we previously used in Figure 4.9 to illustrate the discrete version of first-order regularization, shows an $\mathcal{N}_4$ MRF. The $s_x(i, j)$ and $s_y(i, j)$ black boxes denote arbitrary *interaction potentials* between adjacent nodes in the random field and the $c(i, j)$ denote the data penalty functions. These square nodes can also be interpreted as *factors* in a *factor graph* version of the (undirected) graphical model (Bishop 2006; Dellaert and Kaess 2017; Dellaert 2021), which is another name for interaction potentials. (Strictly speaking, the factors are (improper) probability functions whose product is the (un-normalized) posterior distribution.)

As we will see in (4.41–4.42), there is a close relationship between these interaction potentials and the discretized versions of regularized image restoration problems. Thus, to a first approximation, we can view energy minimization being performed when solving a regularized problem and the maximum *a posteriori* inference being performed in an MRF as equivalent.

While $\mathcal{N}_4$ neighborhoods are most commonly used, in some applications $\mathcal{N}_8$ (or even higher order) neighborhoods perform better at tasks such as image segmentation because they can better model discontinuities at different orientations (Boykov and Kolmogorov 2003; Rother, Kohli *et al.* 2009; Kohli, Ladický, and Torr 2009; Kohli, Kumar, and Torr 2009).

Binary MRFs

The simplest possible example of a Markov random field is a binary field. Examples of such fields include 1-bit (black and white) scanned document images as well as images segmented into foreground and background regions.

To denoise a scanned image, we set the data penalty to reflect the agreement between the scanned and final images,

$$E_{\mathrm{D}}(i, j) = w\delta(f(i, j), d(i, j)) \tag{4.39}$$

and the smoothness penalty to reflect the agreement between neighboring pixels

$$E_{\mathrm{P}}(i,j) = s\delta(f(i,j), f(i+1,j)) + s\delta(f(i,j), f(i,j+1)). \tag{4.40}$$

Once we have formulated the energy, how do we minimize it? The simplest approach is to perform gradient descent, flipping one state at a time if it produces a lower energy. This approach is known as *contextual classification* (Kittler and Föglein 1984), *iterated conditional modes* (ICM) (Besag 1986), or *highest confidence first* (HCF) (Chou and Brown 1990) if the pixel with the largest energy decrease is selected first.

Unfortunately, these downhill methods tend to get easily stuck in local minima. An alternative approach is to add some randomness to the process, which is known as *stochastic gradient descent* (Metropolis, Rosenbluth *et al.* 1953; Geman and Geman 1984). When the amount of noise is decreased over time, this technique is known as *simulated annealing* (Kirkpatrick, Gelatt, and Vecchi 1983; Carnevali, Coletti, and Patarnello 1985; Wolberg and Pavlidis 1985; Swendsen and Wang 1987) and was first popularized in computer vision by Geman and Geman (1984) and later applied to stereo matching by Barnard (1989), among others.

Even this technique, however, does not perform that well (Boykov, Veksler, and Zabih 2001). For binary images, a much better technique, introduced to the computer vision community by Boykov, Veksler, and Zabih (2001) is to re-formulate the energy minimization as a *max-flow/min-cut* graph optimization problem (Greig, Porteous, and Seheult 1989). This technique has informally come to be known as *graph cuts* in the computer vision community (Boykov and Kolmogorov 2011). For simple energy functions, e.g., those where the penalty for non-identical neighboring pixels is a constant, this algorithm is guaranteed to produce the *global minimum*. Kolmogorov and Zabih (2004) formally characterize the class of binary energy potentials (*regularity conditions*) for which these results hold, while newer work by Komodakis, Tziritas, and Paragios (2008) and Rother, Kolmogorov *et al.* (2007) provide good algorithms for the cases when they do not, i.e., for energy functions that are not *regular* or *sub-modular*.

In addition to the above mentioned techniques, a number of other optimization approaches have been developed for MRF energy minimization, such as (loopy) belief propagation and dynamic programming (for one-dimensional problems). These are discussed in more detail in Appendix B.5 as well as the comparative survey papers by Szeliski, Zabih *et al.* (2008) and Kappes, Andres *et al.* (2015), which have associated benchmarks and code at https://vision.middlebury.edu/MRF and http://hciweb2.iwr.uni-heidelberg.de/opengm.

Ordinal-valued MRFs

In addition to binary images, Markov random fields can be applied to ordinal-valued labels such as grayscale images or depth maps. The term "ordinal" indicates that the labels have an implied ordering, e.g., that higher values are lighter pixels. In the next section, we look at unordered labels, such as source image labels for image compositing.

In many cases, it is common to extend the binary data and smoothness prior terms as

$$E_{\mathrm{D}}(i,j) = c(i,j)\rho_d(f(i,j) - d(i,j)) \tag{4.41}$$

and

$$E_{\mathrm{P}}(i,j) = s_x(i,j)\rho_p(f(i,j) - f(i+1,j)) + s_y(i,j)\rho_p(f(i,j) - f(i,j+1)), \tag{4.42}$$

which are robust generalizations of the quadratic penalty terms (4.26) and (4.24), first introduced in (4.29). As before, the $c(i,j)$, $s_x(i,j)$, and $s_y(i,j)$ weights can be used to locally control the data

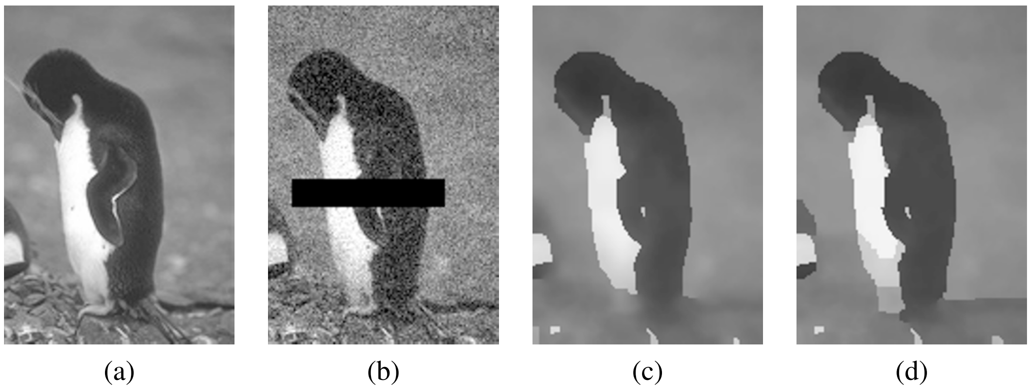


Figure 4.13 Grayscale image denoising and inpainting: (a) original image; (b) image corrupted by noise and with missing data (black bar); (c) image restored using loopy belief propagation; (d) image restored using expansion move graph cuts. Images are from https://vision.middlebury.edu/MRF/results (Szeliski, Zabih *et al.* 2008).

weighting and the horizontal and vertical smoothness. Instead of using a quadratic penalty, however, a general monotonically increasing penalty function $\rho()$ is used. (Different functions can be used for the data and smoothness terms.) For example, ρ_p can be a hyper-Laplacian penalty

$$\rho_p(d) = |d|^p, \quad p < 1, \tag{4.43}$$

which better encodes the distribution of gradients (mainly edges) in an image than either a quadratic or linear (total variation) penalty.[6] Levin and Weiss (2007) use such a penalty to separate a transmitted and reflected image (Figure 9.16) by encouraging gradients to lie in one or the other image, but not both. Levin, Fergus *et al.* (2007) use the hyper-Laplacian as a prior for image deconvolution (deblurring) and Krishnan and Fergus (2009) develop a faster algorithm for solving such problems. For the data penalty, ρ_d can be quadratic (to model Gaussian noise) or the log of a *contaminated Gaussian* (Appendix B.3).

When ρ_p is a quadratic function, the resulting Markov random field is called a Gaussian Markov random field (GMRF) and its minimum can be found by sparse linear system solving (4.28). When the weighting functions are uniform, the GMRF becomes a special case of Wiener filtering (Section 3.4.1). Allowing the weighting functions to depend on the input image (a special kind of conditional random field, which we describe below) enables quite sophisticated image processing algorithms to be performed, including colorization (Levin, Lischinski, and Weiss 2004), interactive tone mapping (Lischinski, Farbman *et al.* 2006), natural image matting (Levin, Lischinski, and Weiss 2008), and image restoration (Tappen, Liu *et al.* 2007).

When ρ_d or ρ_p are non-quadratic functions, gradient descent techniques such as non-linear least squares or iteratively re-weighted least squares can sometimes be used (Appendix A.3). However, if the search space has lots of local minima, as is the case for stereo matching (Barnard 1989; Boykov, Veksler, and Zabih 2001), more sophisticated techniques are required.

The extension of graph cut techniques to multi-valued problems was first proposed by Boykov, Veksler, and Zabih (2001). In their paper, they develop two different algorithms, called the *swap move* and the *expansion move*, which iterate among a series of binary labeling sub-problems to

[6]Note that, unlike a quadratic penalty, the sum of the horizontal and vertical derivative p-norms is not rotationally invariant. A better approach may be to locally estimate the gradient direction and to impose different norms on the perpendicular and parallel components, which Roth and Black (2007b) call a *steerable random field*.

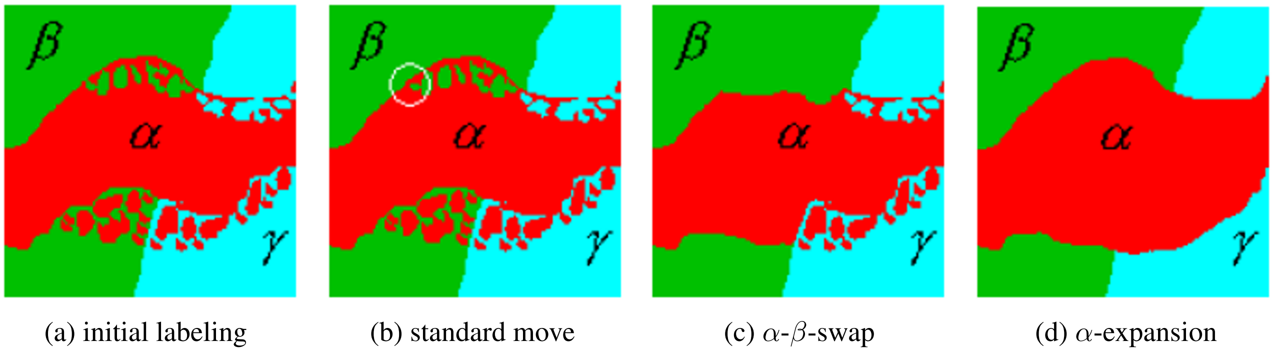


Figure 4.14 Multi-level graph optimization from Boykov, Veksler, and Zabih (2001) © 2001 IEEE: (a) initial problem configuration; (b) the standard move only changes one pixel; (c) the α-β-swap optimally exchanges all α and β-labeled pixels; (d) the α-expansion move optimally selects among current pixel values and the α label.

find a good solution (Figure 4.14). Note that a global solution is generally not achievable, as the problem is provably NP-hard for general energy functions. Because both these algorithms use a binary MRF optimization inside their inner loop, they are subject to the kind of constraints on the energy functions that occur in the binary labeling case (Kolmogorov and Zabih 2004).

Another MRF inference technique is *belief propagation* (BP). While belief propagation was originally developed for inference over trees, where it is exact (Pearl 1988), it has more recently been applied to graphs with loops such as Markov random fields (Freeman, Pasztor, and Carmichael 2000; Yedidia, Freeman, and Weiss 2001). In fact, some of the better performing stereo-matching algorithms use loopy belief propagation (LBP) to perform their inference (Sun, Zheng, and Shum 2003). LBP is discussed in more detail in comparative survey papera on MRF optimization (Szeliski, Zabih *et al.* 2008; Kappes, Andres *et al.* 2015).

Figure 4.13 shows an example of image denoising and inpainting (hole filling) using a non-quadratic energy function (non-Gaussian MRF). The original image has been corrupted by noise and a portion of the data has been removed (the black bar). In this case, the loopy belief propagation algorithm computes a slightly lower energy and also a smoother image than the alpha-expansion graph cut algorithm.

Of course, the above formula (4.42) for the smoothness term $E_P(i,j)$ just shows the simplest case. In follow-on work, Roth and Black (2009) propose a *Field of Experts* (FoE) model, which sums up a large number of exponentiated local filter outputs to arrive at the smoothness penalty. Weiss and Freeman (2007) analyze this approach and compare it to the simpler hyper-Laplacian model of natural image statistics. Lyu and Simoncelli (2009) use *Gaussian Scale Mixtures* (GSMs) to construct an inhomogeneous multi-scale MRF, with one (positive exponential) GMRF modulating the variance (amplitude) of another Gaussian MRF.

It is also possible to extend the *measurement* model to make the sampled (noise-corrupted) input pixels correspond to blends of unknown (latent) image pixels, as in Figure 4.15. This is the commonly occurring case when trying to deblur an image. While this kind of a model is still a traditional generative Markov random field, i.e., we can in principle generate random samples from the prior distribution, finding an optimal solution can be difficult because the clique sizes get larger. In such situations, gradient descent techniques, such as iteratively reweighted least squares, can be used (Joshi, Zitnick *et al.* 2009). Exercise 4.4 has you explore some of these issues.

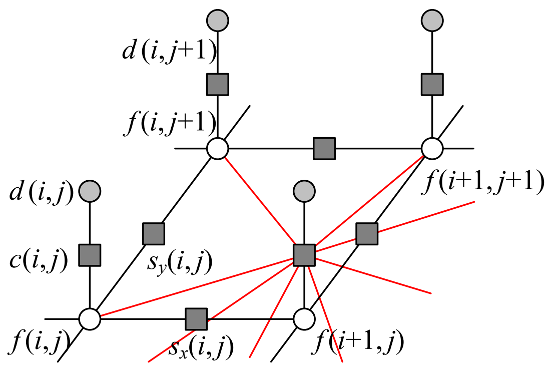


Figure 4.15 Graphical model for a Markov random field with a more complex measurement model. The additional colored edges show how combinations of unknown values (say, in a sharp image) produce the measured values (a noisy blurred image). The resulting graphical model is still a classic MRF and is just as easy to sample from, but some inference algorithms (e.g., those based on graph cuts) may not be applicable because of the increased network complexity, since state changes during the inference become more entangled and the posterior MRF has much larger cliques.

Unordered labels

Another case with multi-valued labels where Markov random fields are often applied is that of *unordered labels*, i.e., labels where there is no semantic meaning to the numerical difference between the values of two labels. For example, if we are classifying terrain from aerial imagery, it makes no sense to take the numerical difference between the labels assigned to forest, field, water, and pavement. In fact, the adjacencies of these various kinds of terrain each have different likelihoods, so it makes more sense to use a prior of the form

$$E_{\rm P}(i,j) = s_x(i,j)V(l(i,j), l(i+1,j)) + s_y(i,j)V(l(i,j), l(i,j+1)), \tag{4.44}$$

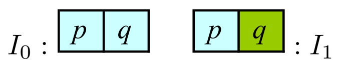
where $V(l_0, l_1)$ is a general *compatibility* or *potential* function. (Note that we have also replaced $f(i,j)$ with $l(i,j)$ to make it clearer that these are labels rather than function samples.) An alternative way to write this prior energy (Boykov, Veksler, and Zabih 2001; Szeliski, Zabih *et al.* 2008) is

$$E_{\rm P} = \sum_{(p,q)\in\mathcal{N}} V_{p,q}(l_p, l_q), \tag{4.45}$$

where the (p,q) are neighboring pixels and a spatially varying potential function $V_{p,q}$ is evaluated for each neighboring pair.

An important application of unordered MRF labeling is seam finding in image compositing (Davis 1998; Agarwala, Dontcheva *et al.* 2004) (see Figure 4.16, which is explained in more detail in Section 8.4.2). Here, the compatibility $V_{p,q}(l_p, l_q)$ measures the quality of the visual appearance that would result from placing a pixel p from image l_p next to a pixel q from image l_q. As with most MRFs, we assume that $V_{p,q}(l,l) = 0$. For different labels, however, the compatibility $V_{p,q}(l_p, l_q)$ may depend on the values of the underlying pixels $I_{l_p}(p)$ and $I_{l_q}(q)$.

Consider, for example, where one image I_0 is all sky blue, i.e., $I_0(p) = I_0(q) = B$, while the other image I_1 has a transition from sky blue, $I_1(p) = B$, to forest green, $I_1(q) = G$.



In this case, $V_{p,q}(1,0) = 0$ (the colors agree), while $V_{p,q}(0,1) > 0$ (the colors disagree).

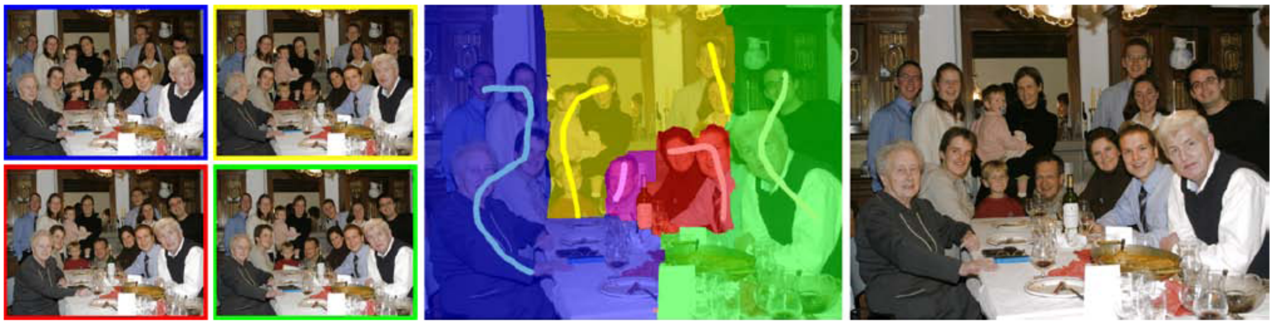


Figure 4.16 An unordered label MRF (Agarwala, Dontcheva *et al.* 2004) © 2004 ACM: Strokes in each of the source images on the left are used as constraints on an MRF optimization, which is solved using graph cuts. The resulting multi-valued label field is shown as a color overlay in the middle image, and the final composite is shown on the right.

4.3.1 Conditional random fields

In a classic Bayesian model (4.33–4.35),

$$p(\mathbf{x}|\mathbf{y}) \propto p(\mathbf{y}|\mathbf{x})p(\mathbf{x}), \tag{4.46}$$

the prior distribution $p(\mathbf{x})$ is independent of the observations $\mathbf{y}$. Sometimes, however, it is useful to modify our prior assumptions, say about the smoothness of the field we are trying to estimate, in response to the sensed data. Whether this makes sense from a probability viewpoint is something we discuss once we have explained the new model.

Consider an interactive image segmentation system such as the one described in Boykov and Funka-Lea (2006). In this application, the user draws foreground and background strokes, and the system then solves a binary MRF labeling problem to estimate the extent of the foreground object. In addition to minimizing a data term, which measures the pointwise similarity between pixel colors and the inferred region distributions (Section 4.3.2), the MRF is modified so that the smoothness terms $s_x(x, y)$ and $s_y(x, y)$ in Figure 4.12 and (4.42) depend on the magnitude of the gradient between adjacent pixels.[7]

Since the smoothness term now depends on the data, Bayes' rule (4.46) no longer applies. Instead, we use a direct model for the posterior distribution $p(\mathbf{x}|\mathbf{y})$, whose negative log likelihood can be written as

$$\begin{aligned} E(\mathbf{x}|\mathbf{y}) &= E_{\mathrm{D}}(\mathbf{x},\mathbf{y}) + E_{\mathrm{S}}(\mathbf{x},\mathbf{y}) \\ &= \sum_p V_p(x_p,\mathbf{y}) + \sum_{(p,q)\in\mathcal{N}} V_{p,q}(x_p,x_q,\mathbf{y}), \end{aligned} \tag{4.47}$$

using the notation introduced in (4.45). The resulting probability distribution is called a *conditional random field* (CRF) and was first introduced to the computer vision field by Kumar and Hebert (2003), based on earlier work in text modeling by Lafferty, McCallum, and Pereira (2001).

Figure 4.17 shows a graphical model where the smoothness terms depend on the data values. In this particular model, each smoothness term depends only on its adjacent pair of data values, i.e., terms are of the form $V_{p,q}(x_p, x_q, y_p, y_q)$ in (4.47).

[7] An alternative formulation that also uses detected edges to modulate the smoothness of a depth or motion field and hence to integrate multiple lower level vision modules is presented by Poggio, Gamble, and Little (1988).

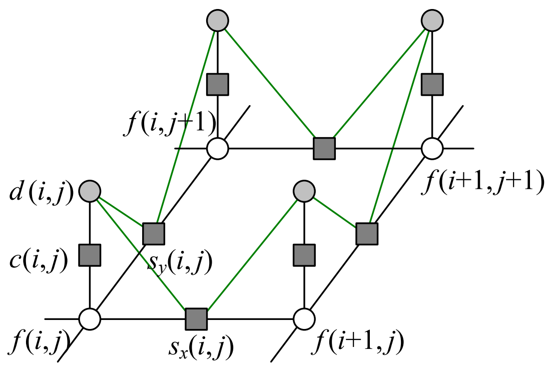


Figure 4.17 Graphical model for a conditional random field (CRF). The additional green edges show how combinations of sensed data influence the smoothness in the underlying MRF prior model, i.e., $s_x(i,j)$ and $s_y(i,j)$ in (4.42) depend on adjacent $d(i,j)$ values. These additional links (factors) enable the smoothness to depend on the input data. However, they make sampling from this MRF more complex.

The idea of modifying smoothness terms in response to input data is not new. For example, Boykov and Jolly (2001) used this idea for interactive segmentation, and it is now widely used in image segmentation (Section 4.3.2) (Blake, Rother *et al.* 2004; Rother, Kolmogorov, and Blake 2004), denoising (Tappen, Liu *et al.* 2007), and object recognition (Section 6.4) (Winn and Shotton 2006; Shotton, Winn *et al.* 2009).

In stereo matching, the idea of encouraging disparity discontinuities to coincide with intensity edges goes back even further to the early days of optimization and MRF-based algorithms (Poggio, Gamble, and Little 1988; Fua 1993; Bobick and Intille 1999; Boykov, Veksler, and Zabih 2001) and is discussed in more detail in (Section 12.5).

In addition to using smoothness terms that adapt to the input data, Kumar and Hebert (2003) also compute a neighborhood function over the input data for each $V_p(x_p, \mathbf{y})$ term, as illustrated in Figure 4.18, instead of using the classic unary MRF data term $V_p(x_p, y_p)$ shown in Figure 4.12.[8] Because such neighborhood functions can be thought of as *discriminant* functions (a term widely used in machine learning (Bishop 2006)), they call the resulting graphical model a *discriminative random field* (DRF). In their paper, Kumar and Hebert (2006) show that DRFs outperform similar CRFs on a number of applications, such as structure detection and binary image denoising.

Here again, one could argue that previous stereo correspondence algorithms also look at a neighborhood of input data, either explicitly, because they compute correlation measures (Criminisi, Cross *et al.* 2006) as data terms, or implicitly, because even pixel-wise disparity costs look at several pixels in either the left or right image (Barnard 1989; Boykov, Veksler, and Zabih 2001).

What then are the advantages and disadvantages of using conditional or discriminative random fields instead of MRFs?

Classic Bayesian inference (MRF) assumes that the prior distribution of the data is independent of the measurements. This makes a lot of sense: if you see a pair of sixes when you first throw a pair of dice, it would be unwise to assume that they will always show up thereafter. However, if after playing for a long time you detect a statistically significant bias, you may want to adjust your prior. What CRFs do, in essence, is to select or modify the prior model based on observed data. This can be viewed as making a partial inference over additional hidden variables or correlations between the unknowns (say, a label, depth, or clean image) and the knowns (observed images).

[8] Kumar and Hebert (2006) call the unary potentials $V_p(x_p, \mathbf{y})$ *association potentials* and the pairwise potentials $V_{p,q}(x_p, y_q, \mathbf{y})$ *interaction potentials*.

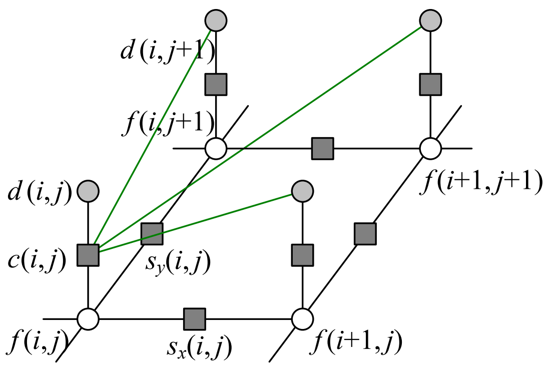


Figure 4.18 Graphical model for a discriminative random field (DRF). The additional green edges show how combinations of sensed data, e.g., $d(i, j+1)$, influence the data term for $f(i, j)$. The generative model is therefore more complex, i.e., we cannot just apply a simple function to the unknown variables and add noise.

In some cases, the CRF approach makes a lot of sense and is, in fact, the only plausible way to proceed. For example, in grayscale image colorization (Section 4.2.4) (Levin, Lischinski, and Weiss 2004), a commonly used way to transfer the continuity information from the input grayscale image to the unknown color image is to modify the local smoothness constraints. Similarly, for simultaneous segmentation and recognition (Winn and Shotton 2006; Shotton, Winn *et al.* 2009), it makes a lot of sense to permit strong color edges to increase the likelihood of semantic image label discontinuities.

In other cases, such as image denoising, the situation is more subtle. Using a non-quadratic (robust) smoothness term as in (4.42) plays a qualitatively similar role to setting the smoothness based on local gradient information in a Gaussian MRF (GMRF) (Tappen, Liu *et al.* 2007; Tanaka and Okutomi 2008). The advantage of Gaussian MRFs, when the smoothness can be correctly inferred, is that the resulting quadratic energy can be minimized in a single step, i.e., by solving a sparse set of linear equations. However, for situations where the discontinuities are not self-evident in the input data, such as for piecewise-smooth sparse data interpolation (Blake and Zisserman 1987; Terzopoulos 1988), classic robust smoothness energy minimization may be preferable. Thus, as with most computer vision algorithms, a careful analysis of the problem at hand and desired robustness and computation constraints may be required to choose the best technique.

Perhaps the biggest advantage of CRFs and DRFs, as argued by Kumar and Hebert (2006), Tappen, Liu *et al.* (2007), and Blake, Rother *et al.* (2004), is that learning the model parameters is more principled and sometimes easier. While learning parameters in MRFs and their variants is not a topic that we cover in this book, interested readers can find more details in publications by Kumar and Hebert (2006), Roth and Black (2007a), Tappen, Liu *et al.* (2007), Tappen (2007), and Li and Huttenlocher (2008).

Dense Conditional Random Fields (CRFs)

As with regular Markov random fields, conditional random fields (CRFs) are normally defined over small neighborhoods, e.g., the $\mathcal{N}_4$ neighborhood shown in Figure 4.17. However, images often contain longer-range interactions, e.g., pixels of similar colors may belong to related classes (Figure 4.19). In order to model such longer-range interactions, Krähenbühl and Koltun (2011) introduced what they call a *fully connected CRF*, which many people now call a *dense CRF*.

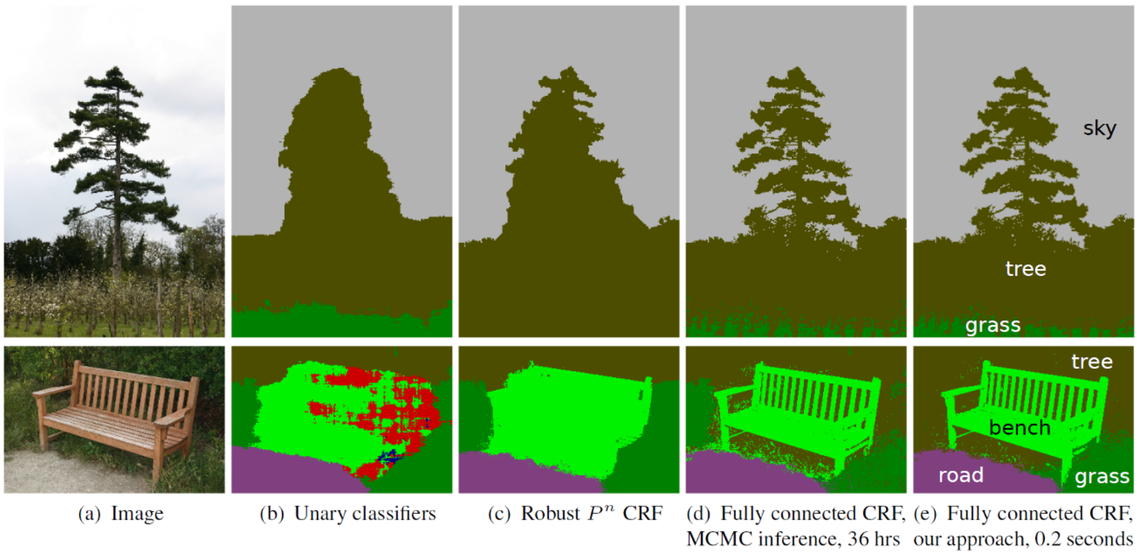

sky
tree
grass
tree
bench
road
grass
(a) Image
(b) Unary classifiers
(c) Robust P^n CRF
(d) Fully connected CRF, MCMC inference, 36 hrs
(e) Fully connected CRF, our approach, 0.2 seconds

Figure 4.19 Pixel-level classification with a fully connected CRF, from © Krähenbühl and Koltun (2011). The labels in each column describe the image or algorithm being run, which include a robust P^n CRF (Kohli, Ladický, and Torr 2009) and a very slow MCMC optimization algorithm.

As with traditional conditional random fields (4.47), their energy function consists of both unary terms and pairwise terms

$$E(\mathbf{x}|\mathbf{y}) = \sum_p V_p(x_p, \mathbf{y}) + \sum_{(p,q)} V_{p,q}(x_p, x_q, y_p, y_q), \tag{4.48}$$

where the (p, q) summation is now taken over *all* pairs of pixels, and not just adjacent ones.[9] The $\mathbf{y}$ denotes the input (guide) image over which the random field is conditioned. The pairwise interaction potentials have a restricted form

$$V_{p,q}(x_p, x_q, y_p, y_q) = \mu(x_p, x_q) \sum_{m=1}^{M} s_m w_m(p, q) \tag{4.49}$$

that is the product of a spatially invariant *label compatibility function* $\mu(x_p, x_q)$ and a sum of M Gaussian kernels of the same form (3.37) as is used in bilateral filtering and the bilateral solver. In their seminal paper, Krähenbühl and Koltun (2011) use two kernels, the first of which is an *appearance kernel* similar to (3.37) and the second is a spatial-only *smoothness kernel*.

Because of the special form of the long-range interaction potentials, which encapsulate all spatial and color similarity terms into a bilateral form, higher-dimensional filtering algorithms similar to those used in fast bilateral filters and solvers (Adams, Baek, and Davis 2010) can be used to efficiently compute a *mean field approximation* to the posterior conditional distribution (Krähenbühl and Koltun 2011). Figure 4.19 shows a comparison of their results (rightmost column) with previous approaches, including using simple unary terms, a robust CRF (Kohli, Ladický, and Torr 2009), and a very slow MCMC (Markov chain Monte Carlo) inference algorithm. As you can see, the fully connected CRF with a mean field solver produces dramatically better results in a very short time.

[9] In practice, as with bilateral filtering and the bilateral solver, the spatial extent may be over a large but finite region.

Since the publication of this paper, provably convergent and more efficient inference algorithms have been developed both by the original authors (Krähenbühl and Koltun 2013) and others (Vineet, Warrell, and Torr 2014; Desmaison, Bunel *et al.* 2016). Dense CRFs have seen widespread use in image segmentation problems and also as a "clean-up" stage for deep neural networks, as in the widely cited DeepLab paper by Chen, Papandreou *et al.* (2018).

4.3.2 *Application*: Interactive segmentation

The goal of image segmentation algorithms is to group pixels that have similar appearance (statistics) and to have the boundaries between pixels in different regions be of short length and across visible discontinuities. If we restrict the boundary measurements to be between immediate neighbors and compute region membership statistics by summing over pixels, we can formulate this as a classic pixel-based energy function using either a *variational formulation* (Section 4.2) or as a binary Markov random field (Section 4.3).

Examples of the continuous approach include Mumford and Shah (1989), Chan and Vese (2001), Zhu and Yuille (1996), and Tabb and Ahuja (1997) along with the level set approaches discussed in Section 7.3.2. An early example of a discrete labeling problem that combines both region-based and boundary-based energy terms is the work of Leclerc (1989), who used minimum description length (MDL) coding to derive the energy function being minimized. Boykov and Funka-Lea (2006) present a wonderful survey of various energy-based techniques for binary object segmentation, some of which we discuss below.

As we saw earlier in this chapter, the energy corresponding to a segmentation problem can be written (c.f. Equations (4.24) and (4.35–4.42)) as

$$E(f) = \sum_{i,j} E_{\text{R}}(i,j) + E_{\text{P}}(i,j), \tag{4.50}$$

where the region term

$$E_{\text{R}}(i,j) = C(I(i,j); R(f(i,j))) \tag{4.51}$$

is the negative log likelihood that pixel intensity (or color) $I(i,j)$ is consistent with the statistics of region $R(f(i,j))$ and the boundary term

$$E_{\text{P}}(i,j) = s_x(i,j)\delta(f(i,j), f(i+1,j)) + s_y(i,j)\delta(f(i,j), f(i,j+1)) \tag{4.52}$$

measures the inconsistency between $\mathcal{N}_4$ neighbors modulated by local horizontal and vertical smoothness terms $s_x(i,j)$ and $s_y(i,j)$.

Region statistics can be something as simple as the mean gray level or color (Leclerc 1989), in which case

$$C(I; \mu_k) = \|I - \mu_k\|^2. \tag{4.53}$$

Alternatively, they can be more complex, such as region intensity histograms (Boykov and Jolly 2001) or color Gaussian mixture models (Rother, Kolmogorov, and Blake 2004). For smoothness (boundary) terms, it is common to make the strength of the smoothness $s_x(i,j)$ inversely proportional to the local edge strength (Boykov, Veksler, and Zabih 2001).

Originally, energy-based segmentation problems were optimized using iterative gradient descent techniques, which were slow and prone to getting trapped in local minima. Boykov and Jolly (2001) were the first to apply the binary MRF optimization algorithm developed by Greig, Porteous, and Seheult (1989) to binary object segmentation.

In this approach, the user first delineates pixels in the background and foreground regions using a few strokes of an image brush. These pixels then become the *seeds* that tie nodes in the *S–T graph*

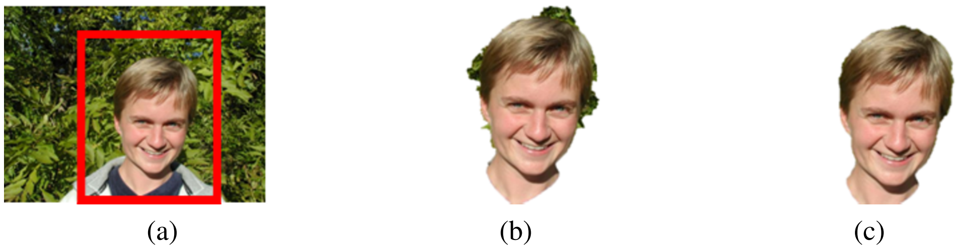


Figure 4.20 GrabCut image segmentation (Rother, Kolmogorov, and Blake 2004) © 2004 ACM: (a) the user draws a bounding box in red; (b) the algorithm guesses color distributions for the object and background and performs a binary segmentation; (c) the process is repeated with better region statistics.

to the source and sink labels S and T. Seed pixels can also be used to estimate foreground and background region statistics (intensity or color histograms).

The capacities of the other edges in the graph are derived from the region and boundary energy terms, i.e., pixels that are more compatible with the foreground or background region get stronger connections to the respective source or sink; adjacent pixels with greater smoothness also get stronger links. Once the minimum-cut/maximum-flow problem has been solved using a polynomial time algorithm (Goldberg and Tarjan 1988; Boykov and Kolmogorov 2004), pixels on either side of the computed cut are labeled according to the source or sink to which they remain connected. While graph cuts is just one of several known techniques for MRF energy minimization, it is still the one most commonly used for solving binary MRF problems.

The basic binary segmentation algorithm of Boykov and Jolly (2001) has been extended in a number of directions. The *GrabCut* system of Rother, Kolmogorov, and Blake (2004) iteratively re-estimates the region statistics, which are modeled as a mixtures of Gaussians in color space. This allows their system to operate given minimal user input, such as a single bounding box (Figure 4.20a)—the background color model is initialized from a strip of pixels around the box outline. (The foreground color model is initialized from the interior pixels, but quickly converges to a better estimate of the object.) The user can also place additional strokes to refine the segmentation as the solution progresses. Cui, Yang *et al.* (2008) use color and edge models derived from previous segmentations of similar objects to improve the local models used in GrabCut. Graph cut algorithms and other variants of Markov and conditional random fields have been applied to the *semantic segmentation* problem (Shotton, Winn *et al.* 2009; Krähenbühl and Koltun 2011), an example of which is shown in Figure 4.19 and which we study in more detail in Section 6.4.

Another major extension to the original binary segmentation formulation is the addition of *directed edges*, which allows boundary regions to be oriented, e.g., to prefer light to dark transitions or *vice versa* (Kolmogorov and Boykov 2005). Figure 4.21 shows an example where the directed graph cut correctly segments the light gray liver from its dark gray surround. The same approach can be used to measure the *flux* exiting a region, i.e., the signed gradient projected normal to the region boundary. Combining oriented graphs with larger neighborhoods enables approximating continuous problems such as those traditionally solved using level sets in the globally optimal graph cut framework (Boykov and Kolmogorov 2003; Kolmogorov and Boykov 2005).

More recent developments in graph cut-based segmentation techniques include the addition of connectivity priors to force the foreground to be in a single piece (Vicente, Kolmogorov, and Rother 2008) and shape priors to use knowledge about an object's shape during the segmentation process (Lempitsky and Boykov 2007; Lempitsky, Blake, and Rother 2008).

While optimizing the binary MRF energy (4.50) requires the use of combinatorial optimiza-

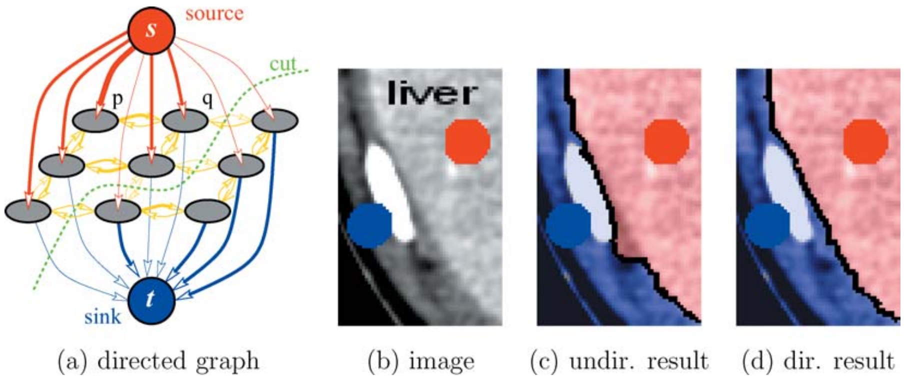


Figure 4.21 Segmentation with a directed graph cut (Boykov and Funka-Lea 2006) © 2006 Springer: (a) directed graph; (b) image with seed points; (c) the undirected graph incorrectly continues the boundary along the bright object; (d) the directed graph correctly segments the light gray region from its darker surround.

tion techniques, such as maximum flow, an approximate solution can be obtained by converting the binary energy terms into quadratic energy terms defined over a continuous $[0, 1]$ random field, which then becomes a classical membrane-based regularization problem (4.24–4.27). The resulting quadratic energy function can then be solved using standard linear system solvers (4.27–4.28), although if speed is an issue, you should use multigrid or one of its variants (Appendix A.5). Once the continuous solution has been computed, it can be thresholded at 0.5 to yield a binary segmentation.

The $[0, 1]$ continuous optimization problem can also be interpreted as computing the probability at each pixel that a *random walker* starting at that pixel ends up at one of the labeled seed pixels, which is also equivalent to computing the potential in a resistive grid where the resistors are equal to the edge weights (Grady 2006; Sinop and Grady 2007). K-way segmentations can also be computed by iterating through the seed labels, using a binary problem with one label set to 1 and all the others set to 0 to compute the relative membership probabilities for each pixel. In follow-on work, Grady and Ali (2008) use a precomputation of the eigenvectors of the linear system to make the solution with a novel set of seeds faster, which is related to the Laplacian matting problem presented in Section 10.4.3 (Levin, Acha, and Lischinski 2008). Couprie, Grady *et al.* (2009) relate the random walker to watersheds and other segmentation techniques. Singaraju, Grady, and Vidal (2008) add directed-edge constraints in order to support flux, which makes the energy piecewise quadratic and hence not solvable as a single linear system. The random walker algorithm can also be used to solve the Mumford–Shah segmentation problem (Grady and Alvino 2008) and to compute fast multigrid solutions (Grady 2008). A nice review of these techniques is given by Singaraju, Grady *et al.* (2011).

An even faster way to compute a continuous $[0, 1]$ approximate segmentation is to compute *weighted geodesic distances* between the 0 and 1 seed regions (Bai and Sapiro 2009), which can also be used to estimate soft alpha mattes (Section 10.4.3). A related approach by Criminisi, Sharp, and Blake (2008) can be used to find fast approximate solutions to general binary Markov random field optimization problems.

4.4 Additional reading

Scattered data interpolation and approximation techniques are fundamental to many different branches of applied mathematics. Some good introductory texts and articles include Amidror (2002), Wendland (2004), and Anjyo, Lewis, and Pighin (2014). These techniques are also related to geometric modeling techniques in computer graphics, which continues to be a very active research area. A nice introduction to basic spline techniques for curves and surfaces can be found in Farin (2002), while more recent approaches using subdivision surfaces are covered in Peters and Reif (2008).

Data interpolation and approximation also lie at the heart of *regression techniques*, which form the mathematical basis for most of the machine learning techniques we study in the next chapter. You can find good introductions to this topic (as well as underfitting, overfitting, and model selection) in texts on classic machine learning (Bishop 2006; Hastie, Tibshirani, and Friedman 2009; Murphy 2012; Deisenroth, Faisal, and Ong 2020) and deep learning (Goodfellow, Bengio, and Courville 2016; Glassner 2018; Zhang, Lipton *et al.* 2021).

Robust data fitting is also central to most computer vision problems. While introduced in this chapter, it is also revisited in Appendix B.3. Classic textbooks and articles on robust fitting and statistics include Huber (1981), Hampel, Ronchetti *et al.* (1986), Black and Rangarajan (1996), Rousseeuw and Leroy (1987), and Stewart (1999). The recent paper by Barron (2019) unifies many of the commonly used robust potential functions and shows how they can be used in machine learning applications.

The regularization approach to computer vision problems was first introduced to the vision community by Poggio, Torre, and Koch (1985) and Terzopoulos (1986a,b, 1988) and continues to be a popular framework for formulating and solving low-level vision problems (Ju, Black, and Jepson 1996; Nielsen, Florack, and Deriche 1997; Nordström 1990; Brox, Bruhn *et al.* 2004; Levin, Lischinski, and Weiss 2008). More detailed mathematical treatment and additional applications can be found in the applied mathematics and statistics literature (Tikhonov and Arsenin 1977; Engl, Hanke, and Neubauer 1996).

Variational formulations have been extensively used in low-level computer vision tasks, including optical flow (Horn and Schunck 1981; Nagel and Enkelmann 1986; Black and Anandan 1993; Alvarez, Weickert, and Sánchez 2000; Brox, Bruhn *et al.* 2004; Zach, Pock, and Bischof 2007a; Wedel, Cremers *et al.* 2009; Werlberger, Pock, and Bischof 2010), segmentation (Kass, Witkin, and Terzopoulos 1988; Mumford and Shah 1989; Caselles, Kimmel, and Sapiro 1997; Paragios and Deriche 2000; Chan and Vese 2001; Osher and Paragios 2003; Cremers 2007), denoising (Rudin, Osher, and Fatemi 1992), stereo (Pock, Schoenemann *et al.* 2008), multi-view stereo (Faugeras and Keriven 1998; Yezzi and Soatto 2003; Pons, Keriven, and Faugeras 2007; Labatut, Pons, and Keriven 2007; Kolev, Klodt *et al.* 2009), and scene flow (Wedel, Brox *et al.* 2011).

The literature on Markov random fields is truly immense, with publications in related fields such as optimization and control theory of which few vision practitioners are even aware. A good guide to the latest techniques is the book edited by Blake, Kohli, and Rother (2011). Other articles that contain nice literature reviews or experimental comparisons include Boykov and Funka-Lea (2006), Szeliski, Zabih *et al.* (2008), Kumar, Veksler, and Torr (2011), and Kappes, Andres *et al.* (2015). MRFs are just one version of the more general topic of graphical models, which is covered in several textbooks and survey, including Bishop (2006, Chapter 8), Koller and Friedman (2009), Nowozin and Lampert (2011), and Murphy (2012, Chapters 10, 17, 19)).

The seminal paper on Markov random fields is the work of Geman and Geman (1984), who introduced this formalism to computer vision researchers and also introduced the notion of *line processes*, additional binary variables that control whether smoothness penalties are enforced or not. Black and Rangarajan (1996) showed how independent line processes could be replaced with robust pairwise potentials; Boykov, Veksler, and Zabih (2001) developed iterative binary graph cut algorithms for optimizing multi-label MRFs; Kolmogorov and Zabih (2004) characterized the class of binary energy potentials required for these techniques to work; and Freeman, Pasztor, and Carmichael (2000) popularized the use of loopy belief propagation for MRF inference. Many more additional references can be found in Sections 4.3 and 4.3.2, and Appendix B.5.

Continuous-energy-based (variational) approaches to interactive segmentation include Leclerc (1989), Mumford and Shah (1989), Chan and Vese (2001), Zhu and Yuille (1996), and Tabb and Ahuja (1997). Discrete variants of such problems are usually optimized using binary graph cuts or other combinatorial energy minimization methods (Boykov and Jolly 2001; Boykov and Kolmogorov 2003; Rother, Kolmogorov, and Blake 2004; Kolmogorov and Boykov 2005; Cui, Yang *et al.* 2008; Vicente, Kolmogorov, and Rother 2008; Lempitsky and Boykov 2007; Lempitsky, Blake, and Rother 2008), although continuous optimization techniques followed by thresholding can also be used (Grady 2006; Grady and Ali 2008; Singaraju, Grady, and Vidal 2008; Criminisi, Sharp, and Blake 2008; Grady 2008; Bai and Sapiro 2009; Couprie, Grady *et al.* 2009). Boykov and Funka-Lea (2006) present a good survey of various energy-based techniques for binary object segmentation.

4.5 Exercises

Ex 4.1: Data fitting (scattered data interpolation). Generate some random samples from a smoothly varying function and then implement and evaluate one or more data interpolation techniques.

1. Generate a "random" 1-D or 2-D function by adding together a small number of sinusoids or Gaussians of random amplitudes and frequencies or scales.

2. Sample this function at a few dozen random locations.

3. Fit a function to these data points using one or more of the scattered data interpolation techniques described in Section 4.1.

4. Measure the fitting error between the estimated and original functions at some set of location, e.g., on a regular grid or at different random points.

5. Manually adjust any parameters your fitting algorithm may have to minimize the output sample fitting error, or use an automated technique such as cross-validation.

6. Repeat this exercise with a new set of random input sample and output sample locations. Does the optimal parameter change, and if so, by how much?

7. (Optional) Generate a piecewise-smooth test function by using different random parameters in different parts of of your image. How much more difficult does the data fitting problem become? Can you think of ways you might mitigate this?

Try to implement your algorithm in NumPy (or Matlab) using only array operations, in order to become more familiar with data-parallel programming and the linear algebra operators built into these systems. Use data visualization techniques such as those in Figures 4.3–4.6 to debug your algorithms and illustrate your results.

Ex 4.2: Graphical model optimization. Download and test out the software on the OpenGM2 library and benchmarks web site http://hciweb2.iwr.uni-heidelberg.de/opengm (Kappes, Andres *et al.* 2015). Try applying these algorithms to your own problems of interest (segmentation, de-noising, etc.). Which algorithms are more suitable for which problems? How does the quality compare to deep learning based approaches, which we study in the next chapter?

Ex 4.3: Image deblocking—challenging. Now that you have some good techniques to distinguish signal from noise, develop a technique to remove the *blocking artifacts* that occur with JPEG at high compression settings (Section 2.3.3). Your technique can be as simple as looking for unexpected edges along block boundaries, or looking at the quantization step as a projection of a convex region of the transform coefficient space onto the corresponding quantized values.

1. Does the knowledge of the compression factor, which is available in the JPEG header information, help you perform better deblocking? See Ehrlich, Lim *et al.* (2020) for a recent paper on this topic.

2. Because the quantization occurs in the DCT transformed YCbCr space (2.116), it may be preferable to perform the analysis in this space. On the other hand, image priors make more sense in an RGB space (or do they?). Decide how you will approach this dichotomy and discuss your choice.

3. While you are at it, since the YCbCr conversion is followed by a chrominance subsampling stage (before the DCT), see if you can restore some of the lost high-frequency chrominance signal using one of the better restoration techniques discussed in this chapter.

4. If your camera has a RAW + JPEG mode, how close can you come to the noise-free true pixel values? (This suggestion may not be that useful, since cameras generally use reasonably high quality settings for their RAW + JPEG models.)

Ex 4.4: Inference in deblurring—challenging. Write down the graphical model corresponding to Figure 4.15 for a non-blind image deblurring problem, i.e., one where the blur kernel is known ahead of time.

What kind of efficient inference (optimization) algorithms can you think of for solving such problems?