Advances in Computer Vision and Pattern Recognition

Marco Alexander Treiber

Optimization for Computer Vision

An Introduction to Core Concepts and Methods

Springer

Advances in Computer Vision and Pattern Recognition

Marco Alexander Treiber

Optimization for Computer Vision

An Introduction to Core Concepts and Methods

Springer

Marco Alexander Treiber
ASM Assembly Systems GmbH & Co. KG
Munich, Germany

Series Editors
Prof. Sameer Singh
Research School of Informatics
Loughborough University
Loughborough
UK

Dr. Sing Bing Kang
Microsoft Research
Microsoft Corporation
Redmond, WA
USA

ISSN 2191-6586 ISSN 2191-6594 (electronic)
Advances in Computer Vision and Pattern Recognition
ISBN 978-1-4471-5282-8 ISBN 978-1-4471-5283-5 (eBook)
DOI 10.1007/978-1-4471-5283-5
Springer London Heidelberg New York Dordrecht

Library of Congress Control Number: 2013943987

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

This book is dedicated to my family:
My parents Maria and Armin
My wife Birgit
My children Lilian and Marisa
I will always carry you in my heart

Preface

In parallel to the much-quoted enduring increase of processing power, we can notice that that the effectiveness of the computer vision algorithms themselves is enhanced steadily. As a consequence, more and more real-world problems can be tackled by computer vision. Apart from their traditional utilization in industrial applications, progress in the field of object recognition and tracking, 3D scene reconstruction, biometrics, etc. leads to a wide-spread usage of computer vision algorithms in applications such as access control, surveillance systems, advanced driver assistance systems, or virtual reality systems, just to name a few.

If someone wants to study this exciting and rapidly developing field of computer vision, he or she probably will observe that many publications primarily focus on the vision algorithms themselves, i.e. their main ideas, their derivation, their performance compared to alternative approaches, and so on.

Compared to that, many contributions place less weight on the rather "technical" issue of the methods of optimization these algorithms employ. However, this does not come up to the actual importance optimization plays in the field of computer vision. First, the vast majority of computer vision algorithms utilize some form of optimization scheme as the task often is to find a solution which is "best" in some respect. Second, the choice of the optimization method seriously affects the performance of the overall method, in terms of accuracy/quality of the solution as well as in terms of runtime. Reason enough for taking a closer look at the field of optimization.

This book is intended for persons being about to familiarize themselves with the field of computer vision as well as for practitioners seeking for knowledge how to implement a certain method. With existing literature, I feel that there are the following shortcomings for those groups of persons:

- The original articles of the computer vision algorithms themselves often don't spend much room on the kind of optimization scheme they employ (as it is assumed that readers already are familiar with it) and often confine themselves at reporting the impact of optimization on the performance.

- General-purpose optimization books give a good overview, but of course lack in relation to computer vision and its specific requirements.
- Dedicated literature dealing with optimization methods used in computer vision often focusses on a specific topic, like graph cuts, etc.

In contrast to that, this book aims at

- Giving a comprehensive overview of a large variety of topics of relevance in computer vision-related optimization. The included material ranges from classical iterative multidimensional optimization to up-to-date topics like graph cuts or GPU-suited total variation-based optimization.
- Bridging the gap between the computer vision applications and the optimization methods being employed.
- Facilitating understanding by focusing on the main ideas and giving (hopefully) clearly written and easy to follow explanations.
- Supplying detailed information how to implement a certain method, such as pseudocode implementations, which are included for most of the methods.

As the main purpose of this book is to introduce into the field of optimization, the content is roughly structured according to a classification of optimization methods (i.e. continuous, variational, and discrete optimization). In order to intensify the understanding of these methods, one or more important example applications in computer vision are presented directly after the corresponding optimization method, such that the reader can immediately learn more about the utilization of the optimization method at hand in computer vision. As a side effect, the reader is introduced into many methods and concepts commonly used in computer vision as well.

Besides hopefully giving easy to follow explanations, the understanding is intended to be facilitated by regarding each method from multiple points of view. Flowcharts should help to get an overview of the proceeding at a coarse level, whereas pseudocode implementations ought to give more detailed insights. Please note, however, that both of them might slightly deviate from the actual implementation of a method in some details for clarity reasons.

To my best knowledge, there does not exist an alternative publication which unifies all of these points. With this material at hand, the interested reader hopefully finds easy to follow information in order to enlarge his knowledge and develop a solid basis of understanding of the field.

Dachau Marco Alexander Treiber
March 2013

Contents

Chapter 1
Introduction

Abstract The vast majority of computer vision algorithms use some form of optimization, as they intend to find some solution which is "best" according to some criterion. Consequently, the field of optimization is worth studying for everyone being seriously interested in computer vision. In this chapter, some expressions being of widespread use in literature dealing with optimization are clarified first. Furthermore, a classification framework is presented, which intends to categorize optimization methods into the four categories continuous, discrete, combinatorial, and variational, according to the nature of the set from which they select their solution. This categorization helps to obtain an overview of the topic and serves as a basis for the structure of the remaining chapters at the same time. Additionally, some concepts being quite common in optimization and therefore being used in diverse applications are presented. Especially to mention are so-called energy functionals measuring the quality of a particular solution by calculating a quantity called "energy", graphs, and last but not least Markov Random Fields.

1.1 Characteristics of Optimization Problems

Optimization plays an important role in computer vision, because many computer vision algorithms employ an optimization step at some point of their proceeding. Before taking a closer look at the diverse optimization methods and their utilization in computer vision, let's first clarify the concept of optimization. Intuitively, in optimization we have to find a solution for a given problem which is "best" in the sense of a certain criterion.

Consider a satnav system, for example: here the satnav has to find the "best" route to a destination location. In order to rate alternative solutions and eventually find out which solution is "best," a suitable criterion has to be applied. A reasonable criterion could be the length of the routes. We then would expect the optimization algorithm to select the route of shortest length as a solution. Observe, however, that other criteria are possible, which might lead to different “optimal” solutions, e.g., the time it takes to travel the route leading to the fastest route as a solution.

Mathematically speaking, optimization can be described as follows: Given a function $f : S \rightarrow \mathbb{R}$ which is called the *objective function*, find the argument x^* which minimizes f:

$$x^* = \arg\min_{x \in S} f(x) \tag{1.1}$$

S defines the so-called solution set, which is the set of all possible solutions for our optimization problem. Sometimes, the unknown(s) x are referred to *design variables*. The function f describes the optimization criterion, i.e., enables us to calculate a quantity which indicates the “goodness” of a particular x.

In the satnav example, S is composed of the roads, streets, motorways, etc., stored in the database of the system, x^* is the route the system has to find, and the optimization criterion $f(x)$ (which measures the optimality of a possible solution) could calculate the travel time or distance to the destination (or a combination of both), depending on our preferences.

Sometimes there also exist one or more additional constraints which the solution x^* has to satisfy. In that case we talk about *constrained optimization* (opposed to *unconstrained optimization* if no such constraint exists). Referring to the satnav example, constraints could be that the route has to pass through a certain location or that we don’t want to use toll roads.

As a summary, an optimization problem has the following “components”:

- One or more design variables x for which a solution has to be found
- An objective function $f(x)$ describing the optimization criterion
- A solution set S specifying the set of possible solutions x
- (optional) One or more constraints on x

In order to be of practical use, an optimization algorithm has to find a solution in a reasonable amount of time with reasonable accuracy. Apart from the performance of the algorithm employed, this also depends on the problem at hand itself. If we can hope for a numerical solution, we say that the problem is *well-posed*. For assessing whether an optimization problem can be solved numerically with reasonable accuracy, the French mathematician Hadamard established several conditions which have to be fulfilled for well-posed problems:

1. A solution exists.
2. There is only one solution to the problem, i.e., the solution is *unique*.
3. The relationship between the solution and the initial conditions is such that small perturbations of the initial conditions result in only small variations of x^*.

If one or more of these conditions is not fulfilled, the problem is said to be *ill-posed*. If condition (3) is not fulfilled, we also speak of *ill-conditioned* problems.

Observe that in computer vision, we often have to solve so-called inverse problems. Consider the relationship $y = T(x)$, for example. Given some kind of observed data y, the inverse problem would be the task to infer a different data representation x from y. The two representations are related via some kind of transformation T. In order to infer x from y directly, we need to know the inverse of T, which explains the name. Please note that this inverse might not exist or could be ambiguous. Hence, inverse problems often are ill-posed problems.

An example of an inverse problem in computer vision is the task of image restoration. Usually, the observed image I is corrupted by noise, defocus, or motion blur. $I(x, y)$ is related to the uncorrupted data R by $I = T(R) + n$, where T could represent some kind of blur (e.g., defocus or motion) and n denotes an additive noise term. Image restoration tries to calculate an estimate of R (termed $\hat{R}$ in the following) from the sensed image I.

A way out of the dilemma of ill-posedness here is to turn the ill-posed problem into a well-posed optimization problem. This can be done by the definition of a so-called energy E, which measures the "goodness" of $\hat{R}$ being an estimation of R. Obviously, E should measure the data fidelity of $\hat{R}$ to I and should be small if $\hat{R}$ doesn't deviate much from I, because usually R is closely related to I. Additional (a priori) knowledge helps to ensure that the optimization problem is well-posed, e.g., we can suppose that the variance of $\hat{R}$ should be as small as possible (because many images contain rather large regions of uniform or slowly varying brightness/color).

In general, the usage of optimization methods in computer vision offers a variety of advantages; among others there are in particular to mention:

- Optimization provides a suitable way of dealing with noise and other sources of corruption.
- The optimization framework enables us to clearly separate between problem formulation (design of the objective function) and finding the solution (employing a suitable algorithm for finding the minimum of the objective function).
- The design of the objective function provides a natural way to incorporate multiple source of information, e.g., by adding multiple terms to the energy function E, each of them capturing a different aspect of the problem.

1.2 Categorization of Optimization Problems

Optimization methods are widely used in numerous computer vision applications of quite diverse nature. As a consequence, the optimization methods which are best suited for a certain application are of quite different nature themselves. However, the optimization methods can be categorized according to their properties. One popular categorization is according to the nature of the solution set S (see e.g. [7]), which will be detailed below.

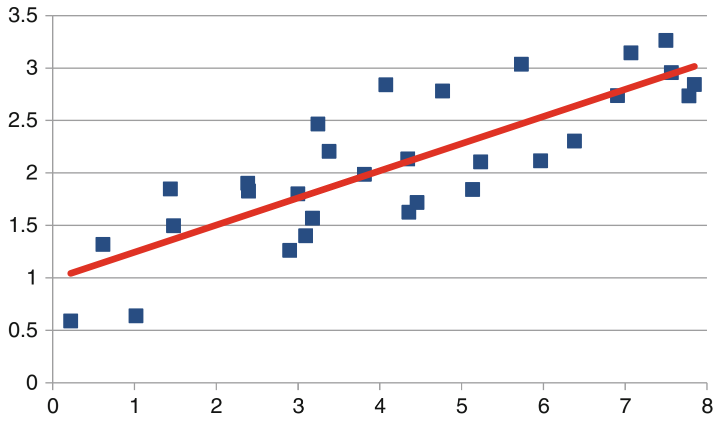

Fig. 1.1 Depicting a set of data points (*blue squares*) and the linear regression line (*bold red line*) minimizing the total sum of squared errors

1.2.1 Continuous Optimization

We talk about *continuous optimization* if the solution set S is a continuous subset of $\mathbb{R}^n$. Typically, this can be a bounded region of $\mathbb{R}^n$, such as a subpixel position $[x, y]$ in a camera image (which is bounded by the image width W and height H: $[x, y] \in [0, \ldots, W-1] \times [0, \ldots, H-1]$) or an m-dimensional subspace of $\mathbb{R}^n$ where m (e.g., a two-dimensional surface of a three-dimensional space – the surface of an object). Here, the bounds or the subspace concept acts as constraints, and these are two examples why continuous optimization methods often have to consider constraints.

A representative application of continuous optimization is *regression*, where observed data shall be approximated by functional relationship. Consider the problem of finding a line that fits to some measured data points $[x_i, y_i]$ in a two-dimensional space (see Fig. 1.1). The line l to be found can be expressed through the functional relationship $l : y = mx + t$. Hence, the problem is to find the parameters m and t of the function. A criterion for the goodness of a particular fit is how close the measured data points are located with respect to the line. Hence, a natural choice for the objective function is a measure of the overall squared distance:

$$f_l(\mathbf{x}) = \sum_i |y_i - (m \cdot x_i + t)|^2$$

Continuous optimization methods directly operate on the objective function and intend to find its minimum numerically. Typically, the objective function $f(\mathbf{x})$ is multidimensional where $\mathbf{x} \in \mathbb{R}^n$ with $n > 1$. As a consequence, the methods often are of iterative nature, where a one-dimensional minimum search along a certain search direction is performed iteratively: starting at an initial solution $\mathbf{x}_{i-1}$, each step i first determines the search direction $\mathbf{a}_i$ and then performs a one-dimensional optimization of $f(\mathbf{x})$ along $\mathbf{a}_i$ yielding an updated solution $\mathbf{x}_i$. The next step repeats this proceeding until convergence is achieved.

A further categorization of continuous optimization methods can be done according to the knowledge of $f(\mathbf{x})$ which is taken into account during optimization: some methods only use knowledge of $f(\mathbf{x})$, some additionally utilize knowledge of its first derivative $\nabla f(\mathbf{x})$ (gradient), and some also make use of its second derivative, i.e., the Hessian matrix $\mathbf{H}(f(\mathbf{x}), \mathbf{x})$.

1.2.2 Discrete Optimization

Discrete optimization deals with problems where the elements of the solution set S take discrete values, e.g., $S \subseteq \mathbb{Z}^n = \{i_1, i_2, \ldots, i_n\}; i_n \in \mathbb{Z}$.

Usually, discrete optimization problems are *NP-hard* to solve, which, informally speaking, in essence states that there is no known algorithm which finds the correct solution in polynomial time. Therefore, execution times soon become infeasible as the size of the problem (the number of unknowns) grows.

As a consequence, many discrete optimization methods aim at finding approximate solutions, which can often be proven to be located within some reasonable bounds to the "true" optimum. These methods are often compared in terms of the quality of the solution they provide, i.e., how close the approximate solution gets to the "true" optimal solution. This is in contrast to continuous optimization problems, which aim at optimizing their rate of convergence to local minima of the objective function.

In practice it turns out that the fact that the solution can only take discrete values, which acts as an additional constraint, often complicates matters when we efficiently want to find a solution. Therefore, a technique called *relaxation* can be applied, where the discrete problem is transformed into its continuous version: The objective function remains unchanged, but now the solution can take continuous values, e.g., by replacing $S_\mathrm{d} \subseteq \mathbb{Z}^n$ with $S_\mathrm{c} \subseteq \mathbb{R}^n$, i.e., the (additional) constraint that the solution has to take discrete values is dropped. The continuous representation can be solved with an appropriate continuous optimization technique. A simple way of deriving the discrete solution x_d^* from the thus obtained continuous one x_c^* is to choose that element of the discrete solution set S_d which is closest to x_c^*. Please note that there is no guarantee that x_d^* is the optimal solution of the discrete problem, but under reasonable conditions it should be sufficiently close to it.

1.2.3 Combinatorial Optimization

In *combinatorial optimization*, the solution set S has a finite number of elements, too. Therefore, any combinatorial optimization problem is also a discrete problem. Additionally, however, for many problems it is impractical to build S as an explicit enumeration of all possible solutions. Instead, a (combinatorial) solution can be expressed as a *combination* of some other representation of the data.

To make things clear, consider to the satnav example again. Here, S is usually not represented by a simple enumeration of all possible routes from the start to a destination location. Instead, the data consists of a map of the roads, streets, motorways, etc., and each route can be obtained by combining these entities (or parts of them). Observe that this allows a much more compact representation of the solution set.

This representation leads to an obvious solution strategy for optimization problems: we "just" have to try all possible combinations and find out which one yields the minimum value of the objective function. Unfortunately, this is infeasible due to the exponential growth of the number of possible solutions when the number of elements to combine increases (a fact which is sometimes called *combinatorial explosion*).

An example of combinatorial optimization methods used in computer vision are the so-called graph cuts, which can, e.g., be utilized in segmentation problems: consider an image showing an object in front of some kind of background. Now we want to obtain a reasonable segmentation of the foreground object from the background. Here, the image can be represented by a graph $G = (V, E)$, where each pixel i is represented by a vertex $v_i \in V$, which is connected to all of its neighbors via an edge $e_{ij} \in E$ (where pixels i and j are adjacent pixels; typically a 4-neighborhood is considered).

A solution s of the segmentation problem which separates the object region from the background consists of a set of edges (where each of these edges connects a pixel located at the border of the object to a background pixel) and can be called a cut of the graph. In order to find the best solution, a cost c_{ij} can be assigned to each edge e_{ij}, which can be derived from the intensity difference between pixel i and j: the higher the intensity difference, the higher c_{ij}. Hence, the solution of the problem is equal to find the cut which minimizes the overall cost along the cut. As each cut defines a combination of edges, graph cuts can be used to solve combinatorial optimization problems. This combinatorial strategy clearly is superior to enumerate all possible segmentations and seek the solution by examination of every element of the enumeration.

1.2.4 Variational Optimization

In *variational optimization,* the solution set S denotes a subspace of *functions* (instead of values in "normal" optimization), i.e., the goal is to find a function which best models some data.

A typical example in computer vision is *image restoration*, where we want to infer an "original" image $R(x, y)$ without perturbations based on a noisy or blurry observation $I(x, y)$ of that image. Hence, the task is to recover the function $R(x, y)$ which models the original image. Consequently, restoration is an example of an inverse problem.

Observe that this problem is ill-posed by nature, mainly because the number of unknowns is larger than the number of observations and therefore many solutions can be aligned with the observation. Typically, we have to estimate $W \times H$ pixel in R and, additionally, some other quantities which model the perturbations like noise variance or a blur kernel, but we have only $W \times H$ observations in I.

Fortunately, such problems can often be converted into a well-posed optimization problem by additionally considering prior knowledge. Based on general considerations, we can often state that some solutions are more likely than others. In this context, the usage of the so-called smoothness assumption is quite common. This means that solutions which are "smooth," i.e., take constant or slowly varying values, are to be favored and considered more likely.

A natural way to consider such a priori information about the solution is the usage of a so-called energy functional E as objective function (a more detailed description is given below in the next section). E measures the "energy" of a particular explanation $\hat{R}$ of the observed data I. If E has low values, $\hat{R}$ should be a good explanation of I. Hence, seeking the minimum of E solves the optimization problem.

In practice E is composed of multiple terms. At first, the so-called external energy models the fidelity of a solution $\hat{R}$ to the observed data I. Obviously, $\hat{R}$ is a good solution if it is close to I. In order to resolve the discrepancy between the number of observed data and unknowns, additional prior knowledge is introduced in E. This term is called *internal energy*. In our example, most natural images contain large areas of uniform or smoothly varying brightness, so a reasonable choice for the internal energy is to integrate all gradients observed in I. As a consequence, the internal energy acts as a regularization term which favors solutions which are in accordance with the smoothness assumption.

To sum it up, the introduction of the internal energy ensures the well-posedness of variational optimization problems. A smoothness constraint is quite common in this context.

Another example are so-called active contours, e.g., "snakes," where the course of a parametric curve has to be estimated. Imagine an image showing an object with intensity distinct from background intensity, but some parts of its boundary cannot be clearly separated from the background. The sought curve should pass along the borders of this object. Accordingly, its external energy is measured by local intensity gradients along the curve. At the same time, object boundaries typically are smooth. Therefore, the internal energy is based on first- and second-order derivatives. These constraints help to fully describe the object boundary by the curve, even at positions where, locally, the object cannot be clearly separated from the background.

A graphical summarization of the four different types of solution sets can be seen in Fig. 1.2.

1.3 Common Optimization Concepts in Computer Vision

Before taking a closer look at the diverse optimization methods, let's first introduce some concepts which are of relevance to optimization and, additionally, in widespread use in computer vision.

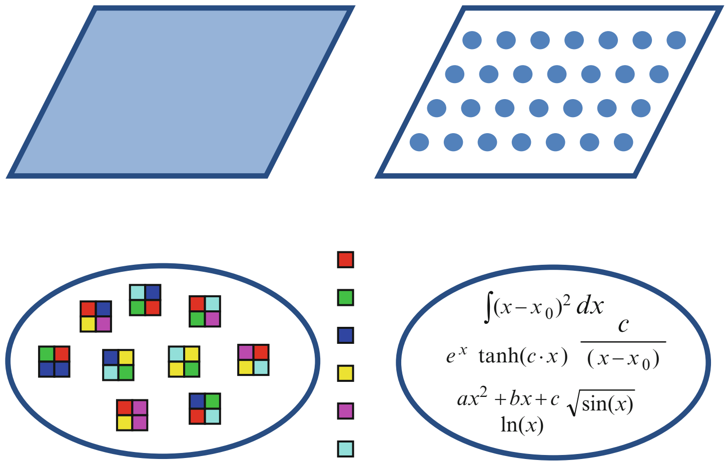

Fig. 1.2 Illustrating the different types of solution sets: continuous (blue two-dimensional region, *upper left*), discrete (grid, *upper right*), combinatorial (combinations of four color squares, *lower left*), and variational (space of functions, *lower right*)

With the help of energy functions, for example, it is possible to evaluate and compare different solutions and thus use this measure to find the optimal solution. The well-known MAP estimator, which finds the best solution by estimating the "most likely" one, given some observed data, can be considered as one form of energy minimization.

Markov Random Fields (MRFs) are a very useful model if the "state" of each pixel (e.g., a label or intensity value) is related to the states of its neighbors, which makes MRFs suitable for restoration (e.g., denoising), segmentation, or stereo-matching tasks, just to name a few.

Last but not least, many computer vision tasks rely on establishing correspondences between two entities. Consider, for example, an object which is represented by a set of characteristic points and their relative position. If such an object has to be detected in a query image, a common proceeding is to extract characteristic points for this image as well and, subsequently, try to match them to the model points, i.e., to establish correspondences between model and query image points.

In addition to this brief explanation, the concepts of energy functions, graphs, and Markov Random Fields are described in more detail in the following sections.

1.3.1 Energy Minimization

The concept of so-called energy functions is a widespread approach in computer vision. In order to find the "best" solution, one reasonable way is to quantify how "good" a particular solution is, because such a measure enables us to compare different solutions and select the "best". Energy functions E are widely used in this context (see, e.g., [6]).

Generally speaking, the "energy" is a measure how plausible a solution is. High energies indicate bad solutions, whereas a low energy signalizes that a particular solution is suitable for explaining some observed data. Some energies are so-called functionals. The term "functional" is used for operators which map a functional relationship to a scalar value (which is the energy here), i.e., take a function as argument (which can, e.g., be discretely represented by a vector of values) and derive a scalar value from this. Functionals are needed in variational optimization, for example.

With the help of such a function, a specific energy can be assigned to each element of the solution space. In this context, optimization amounts to finding the argument which minimizes the function:

$$x^* = \arg\min_{x \in S} E(x) \tag{1.2}$$

As already mentioned in the previous section, E typically consists of two components:

1. A data-driven or external energy E_{ext}, which measures how "good" a solution explains the observed data. In restoration tasks, for example, E_{ext} depends on the fidelity of the reconstructed signal $\hat{R}$ to the observed data I.
2. An internal energy E_{int}, which exclusively depends on the proposed solution (i.e., is independent on the observed data) and quantifies its plausibility. This is the point where a priori knowledge is considered: based on general considerations, we can consider some solutions to be more likely than others and therefore assign a low internal energy to them. In this context it is often assumed that the solution should be "smooth" in a certain sense. In restoration, for example, the proposed solution should contain large areas with uniform or very smoothly varying intensity, and therefore E_{int} depends on some norm of the sum of the gradients between adjacent pixels.

Overall, we can write:

$$E = E_{\text{ext}} + \lambda \cdot E_{\text{int}} \tag{1.3}$$

where the parameter λ specifies the relative weighting between external and internal energy. High values of λ tend to produce smoothly varying optimization results (if E_{int} measures the smoothness of the solution), whereas low values of λ favor results being close to the observed values.

Please observe that the so-called MAP (maximum a posteriori) estimation, which is widely used, too, is closely related to energy minimization. MAP estimation tries to maximize the probability $p(M|D)$ of some model M, given some observed data D ($p(M|D)$ is called the *posterior probability*, because it denotes a probability *after* observing the data). However, it is difficult to quantify $p(M|D)$ in practice. A way out is to apply Bayes' rule:

$$p(M|D) = \frac{p(D|M) \cdot p(M)}{p(D)} \tag{1.4}$$

where $p(D|M)$ is called the *likelihood* of the data and $p(M)$ the *prior*, which measures how probable a certain model is.

If we are only interested in finding the most likely model M^* (and not in the absolute value of the posterior at this position M^*), we can drop $p(D)$ and, additionally, take the logarithm of both sides of (1.4). As a common way to model the probabilities are Gaussian distributions, taking the logarithm simplifies calculations considerably, because it eliminates the exponentials. If we take the negative logarithm, we have:

$$M^* = \arg\min(-\log[p(D|M)] - \log[p(M)]) \tag{1.5}$$

If we set $E_{\text{ext}} = -\log[p(D|M)]$ and $E_{\text{int}} = -\log[p(M)]$, we can see that the structure of (1.5) is the same as we encountered in (1.2).

However, please note that MAP estimation is not completely equivalent to energy-based optimization in every case, as there are many possibilities how to model the terms of the energy functional, and the derivation from MAP estimation is just one kind, albeit a very principled one. Because of this principled proceeding, Bayesian modeling has several potential advantages over user-defined energy functionals, such as:

- The parameters of probability distributions can be *learned* from a set of examples, which in general is more accurate than just estimating or manually setting weights of the energy functional, at least if a suitable training base is available.
- It is possible to estimate complete probability *distributions* over the unknowns instead of determining one single value for each of them at the optimum.
- There exist techniques for optimizing unordered variables (e.g., the labels assigned to different regions in image segmentation tasks) with MAP estimations, whereas unordered variables pose a serious problem when they have to be compared in an energy function.

1.3.2 Graphs

The concept of graphs can be found in various vision applications. In the context of optimization, graphs can be used to model the problem at hand. An optimization procedure being based on a graph model can then utilize specific characteristics of the graph – such as a special structure – in order to get a fast result. In the following,

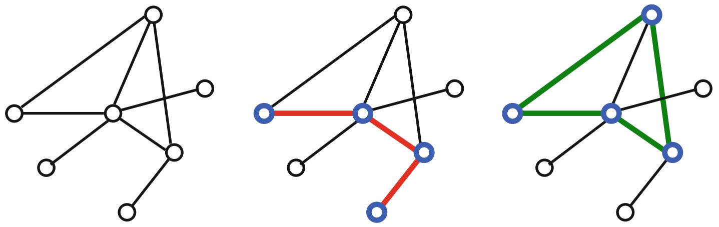

Fig. 1.3 Exemplifying a graph consisting of nodes (*circles*), which are linked by edges (*lines*) (*left*). In the graph shown in the *middle*, the *blue nodes* form a path, which are connected by the *red edges*. *Right*: example of a cycle (*blue nodes*, *green edges*)

some definitions concerning graphs and their properties are introduced (a more detailed introduction can be found in, e.g., [4]).

A *graph* $G = \{N, E\}$ is a set of *nodes* $N = \{n_1, n_2, \ldots, n_L\}$, which are also called *vertices*. The nodes are connected by *edges*, where the edges model the relationship between the nodes: Two nodes n_i and n_j are connected by an edge e_{ij} if they have a special relationship. All edges are pooled in the edge set $E = \{e_{ij}\}$ (see the left of Fig. 1.3 with *circles* as *nodes* and *lines* as *edges* for an example).

Graphs are suitable for modeling a wide variety of computer vision problems. Typically, the nodes model individual pixels or features derived from the image, such as interest points. The edges model the relationship between the pixels and features. For example, an edge e_{ij} indicates that the pixels i and j influence each other. In many cases this influence is limited to a rather small local neighborhood or adjacent pixels. Consequently, edges are confined to nearby or, even more restrictive, adjoining pixels.

Additionally, an edge can feature a direction, i.e., the edge e_{ij} can point from node n_i to node n_j, e.g., because pixel i influences pixel j, but not vice versa. In that case, we talk about a *directed* graph (otherwise the graph is called *undirected*).

Moreover, a weight w_{ij} can be associated to an edge e_{ij}. The weights serve as a measure how strongly two nodes are connected.

A *path* in a graph from node n_i to node n_k denotes a sequence of nodes starting at n_i and ending at n_k, where two consecutive nodes are connected by an edge at a time (see *blue nodes/red edges* in the *middle part* of Fig. 1.3). If n_i is equal to n_k, i.e., start node and termination node are the same, the path is termed a *cycle* (*right part* of Fig. 1.3). The *length* of the path is the sum of the weights of all edges along the path.

An important special case of graphs are trees. A graph is said to be a *tree* if it is undirected, contains no cycles, and, additionally, is connected, which means that there is a path between any two nodes of the graph (see *left* of Fig. 1.4). In a tree, one node can be arbitrarily picked as root node (*light red node* in *right tree* of Fig. 1.4). All other nodes have a certain *depth* from the root, which is related to the number of edges along the path between them and the root (as depth increases, color saturation of the *circles* decreases in the *right part* of Fig. 1.4).

Another subclass of graphs are *bipartite* graphs. A graph is said to be bipartite if its nodes can be split into two disjoint subsets A and B such that any edge of the

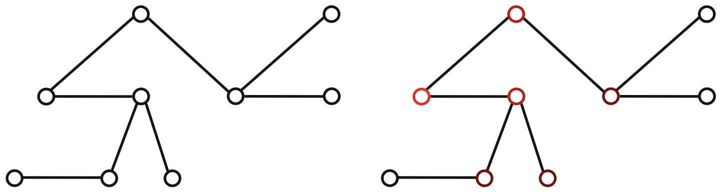

Fig. 1.4 Showing a tree example (*left*). *Right*: same tree, with specification of a root node (*light red*). Depending on their depth, the nodes get increasingly darker

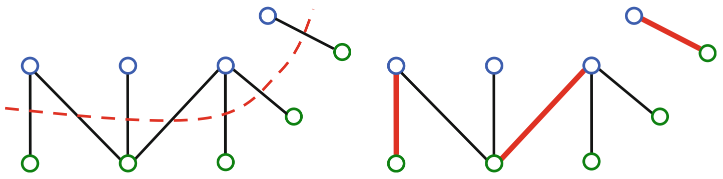

Fig. 1.5 Illustrating a bipartite graph (*left*), where the edges split the nodes into the two *blue* and *green* subsets (separated by *dashed red line*). *Right*: the same bipartite graph, with a matching example (*red edges*)

graph connects one node of A to one node of B. This means that for any edge, exactly one of the nodes it connects is an element of A and exactly one is an element of B (see *left part* of Fig. 1.5, where the *nodes* are split by the *edges* into the *blue* and the *green* subset). Bipartite graphs can be useful for so-called assignment problems, where a set of features has to be matched to another feature set, i.e., for each feature of one set, the “best corresponding” feature of the other set has to be found.

Last but not least, we talk about a *matching* M, if M is a subset of the edge set E with the property that each node of the bipartite graph belongs to at most one edge of M (see *right part* of Fig. 1.5). If each node is connected by exactly one edge $\in M$, M is said to be a *perfect matching*. Observe that the red edge set of the right graph of Fig. 1.5 is not a perfect matching, because some nodes are not connected to any of the red edges.

1.3.3 Markov Random Fields

One example of graphs are so-called Markov Random Fields (or *MRFs*), which are two-dimensional lattices of variables and were introduced for vision applications by [3]. Over the years, they found widespread use in computer vision (see, e.g., [1] for an overview).

In computer vision tasks, it is natural to regard each of those variables as one pixel of an image. Each variable can be interpreted as one node n_i of a graph $G = \{N, E\}$ consisting of a set of nodes N, which are connected by a set of edges E, as described in the previous section. The value each pixel takes is referred to the *state* of the corresponding node.

In practice, the state of each node (pixel) is influenced by other nodes (pixels). This influence is represented in the MRF through edges: if two nodes n_i and n_j influence each other, they are connected by an edge e_{ij}. Without giving the exact mathematical definition here, we note that the nature of MRFs defines that:

- Each node n_i is influenced only by a well-defined neighborhood $S(n_i)$ around it, e.g., 4-neighborhood.
- If $n_j \in S(n_i)$, the relation $n_i \in S(n_j)$ is also true, i.e., if n_j is within the local neighborhood of n_i, then the same observation holds vice versa: n_i is located within the neighborhood of n_j.

A weight s_{ij} can be assigned to each edge e_{ij}. The s_{ij} determines how strong neighboring nodes influence each other and are called *pairwise interaction potentials*. Another characteristic of Markovian models is that a future state of a node depends only on the *current* states of the nodes in the neighborhood. In other words, there is no direct dependency (on past states of them).

Normally, the nodes N represent "hidden" variables, which cannot be measured directly. However, each n_i can be related to a variable o_i which is observable. Each (n_i, o_i) pair is connected by an edge, too. A weight w_i can be assigned to each of these edges as well, and each w_i defines how strong the state of n_i is influenced by o_i.

Markov Random Fields are well suited for Bayesian-modeled energy functionals as introduced in the previous section and in particular for reconstruction or restoration problems: Given a measured image I, which is corrupted by noise, blur, etc., consider the task to reconstruct its uncorrupted version R. In order to represent this problem with an MRF, we make the following assignments:

- Each node (hidden variable) represents one unknown of the optimization problem, i.e., the state of a particular node n_i represents the (unknown) value $R(x_i, y_i)$ of the pixel $[x_i, y_i]$ of the image to be reconstructed.
- Each $R(x_i, y_i)$ is related to the measured value at this position $I(x_i, y_i)$, i.e., each o_i is assigned to one observed value $I(x_i, y_i)$.
- The weight $w(x_i, y_i)$ models how strong $R(x_i, y_i)$ is influenced by the observation $I(x_i, y_i)$. Therefore, high values of w ensure a high fidelity of the R to the measured image data. This corresponds to the relative weighting of the external energy (see parameter λ in (1.3)).
- The edges e_{ij} between the hidden variables can be considered as a way of modeling the influence of the prior probability. One way to do this is to compare the values of neighboring hidden variables (cf. the smoothness assumption, where low intensity differences between adjacent pixels are considered more likely than high differences). In doing so, the weights s_{ij} determine which neighbors influence a particular hidden variable and how strong they do this. Due to its simplicity, it is tempting to use a 4-neighborhood. In many cases, this simple neighborhood proves to be sufficient enough for modeling reality, which makes it the method of choice for many applications.

If we assume a 4-neighborhood, we can model the MRF graphically as demonstrated in Fig. 1.6. The grid of hidden variables, where each variable represents

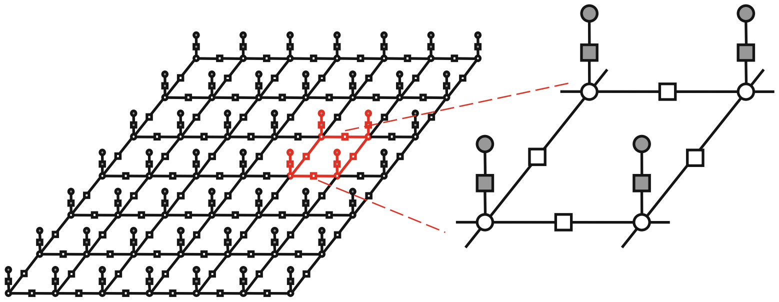

Fig. 1.6 Showing a graphical illustration of a Markov Random Field (*left*) and a more detailed illustration of a part of it (*right*)

a pixel, is shown on the left side of the figure. The right side of the figure illustrates a (zoomed) part of the MRF (red) in more detail. The hidden variables are illustrated via white circles. Each hidden variable is connected to its four neighbors via edges with weights s_x and s_y (symbolized by white squares). An observed data value (gray circles) is associated to each hidden variable with weight w (gray square). It is possible to utilize spatially varying weights $s_x(x, y), s_y(x, y)$, and $w(x, y)$, if desired, but in many cases it is sufficient to use weights being constant over the entire MRF.

If the MRF models an energy functional, the following relationships are frequently used: The external or data-driven energy $E_{\text{ext}}(x, y)$ of each pixel depends on the difference between the state of the hidden variable and the observed value:

$$E_{\text{ext}}(x, y) = w(x, y) \cdot \rho_{\text{ext}}(R(x, y) - I(x, y)) \tag{1.6}$$

The energy being based on the prior follows a smoothness assumption and therefore penalizes adjacent hidden variables of highly differing states:

$$\begin{aligned} E_{\text{int}}(x, y) =& s_x(x, y) \cdot \rho_{\text{int}}(I(x, y) - I(x + 1, y)) + s_y(x, y) \cdot \\ & \rho_{\text{int}}(I(x, y) - I(x, y + 1)) \end{aligned} \tag{1.7}$$

In both (1.6) and (1.7), ρ denotes a monotonically increasing function, e.g., a linear (total variation) penalty $\rho(d) = |d|$ or a quadratic penalty $\rho(d) = |d|^2$. In the case of quadratic penalties, we talk about Gaussian Markov Random Fields (GMRFs), because quadratic penalties are best suited when the corruption of the observed signal is assumed to be of Gaussian nature.

However, in many cases, a significant fraction of measurements is disturbed by outliers with gross errors, and a quadratic weighting puts too much emphasis on the influence of these outliers. Therefore, hyper-Laplacian penalties of the type $\rho(d) = |d|^p;\ p < 1$ are also common.

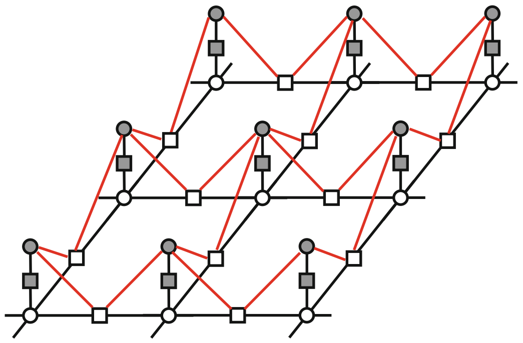

Fig. 1.7 Illustrating the structure of a so-called conditional random field. The influence of the observed data values on the pairwise interaction weights to neighboring nodes is indicated by *red lines*

The total energy associated to the current state of an MRF equals the sum of the external as well as internal energy terms over all pixels:

$$E_{\mathrm{MRF}} = \sum_{x,y} E_{\mathrm{int}}(x,y) + \sum_{x,y} E_{\mathrm{ext}}(x,y) \tag{1.8}$$

Minimization of these types of MRF-based energies used to be performed by a method called *simulated annealing* (cf. [5]) when MRFs were introduced to the vision community (see [3]). Simulated annealing is an iterative scheme which, at each iteration, randomly picks a variable which is to be changed in this iteration. In later stages, the algorithm is rather “greedy,” which means that it has a strong bias toward changes which reduce the energy. In early stages, however, a larger fraction of changes which does not immediately lead to lower energies is allowed. The justification for this is that the possibility to allow changes which increase the energy helps to avoid being “trapped” in a local minimum of the energy functional.

In the meantime, however, it was shown that another class of algorithm called *graph cuts* is better suited for optimization and outperforms simulated annealing in most vision applications (see, e.g., [2]). Here, the algorithm works on the graph representation of the MRF and intends to find a “cut” which separates the graph into two parts where the total sum of the weights of edges to be cut attains a minimum.

MRFs have become popular to model problems where a reasonable prior is to assume that the function to be found varies smoothly. Therefore, it can be hypothesized that each pixel has a high probability to take values identical or very similar to its neighbors, and this can be modeled very well with an MRF. Apart from the already mentioned restoration task, MRFs are also suited for segmentation, because it is unlikely that the segmentation label (all pixels belonging to the same region are “labeled” with identical value) changes frequently between neighboring pixels, which would result in a very fragmented image. The same fact applies for stereo matching tasks, where the disparity between associated pixels, which is a measure of depth of the scene, should vary slowly spatially.

A variant of MRFs are so-called conditional random fields (CRF), which differ from standard MRFs in the influence of the observed data values: More specifically, the weights of the pairwise interaction potentials can be affected by the observed values of the neighboring pixels (see Fig. 1.7). In that case, the prior depends not