ALBERT-LUDWIGS-UNIVERSITÄT
FREIBURG
INSTITUT FÜR INFORMATIK

Lehrstuhl für Mustererkennung und Bildverarbeitung
Prof. Dr. Hans Burkhardt

Efficient Generalized Belief Propagation for Image Segmentation

Diplomarbeit

Kersten Petersen

Mai 2007 – November 2007

Erklärung

Hiermit erkläre ich, daß die vorliegende Arbeit von mir selbständig und nur unter Verwendung der aufgeführten Hilfsmittel erstellt wurde.

Freiburg, den

Danksagung

Zuerst möchte ich Janis Fehr, meinem Betreuer, danken, der mir immer mit Rat und Tat zur Seite stand und mir alle Freiheiten ließ, eigene Ideen zu verfolgen. Ich behalte die angeheme Atmosphäre und die fruchtbaren Diskussionen gerne in Erinnerung.

Großer Dank gebührt auch Prof. Dr. Burkhardt, der mein Interesse für Mustererkennung und Bildverarbeitung geweck hat und mir die Möglichkeit gab, diese Arbeit an seinem Lehrstuhl zu schreiben.

Mario Emmenlauer danke ich für seine tatkräftige Unterstützung bei computertechnischen Problemen, die mich sonst sicherlich einige Nerven gekostet hätten.

Abschließend gilt mein besonderer Dank meinen Eltern, die mich auf unzählige Art und Weise unterstützen, meinen Schwestern Lena und Neele für viele kleine Aufmerksamkeiten und meinen beiden Omas für das eine oder andere “Obstgeld”.

Contents

1 Introduction **5**
- 1.1 Motivation . . . 5
- 1.2 Outline of the thesis . . . 6

2 Image segmentation **7**
- 2.1 Segmentation methods . . . 7
  - 2.1.1 Point-based methods . . . 7
  - 2.1.2 Edge-based methods . . . 8
  - 2.1.3 Region-based methods . . . 9
  - 2.1.4 Model-based methods . . . 9
  - 2.1.5 Discussion . . . 11

3 Probabilistic graphical models **12**
- 3.1 Graphical Models . . . 13
  - 3.1.1 Directed acyclic graphical models . . . 13
  - 3.1.2 Undirected graphical models . . . 14
  - 3.1.3 Factor graphs . . . 14
- 3.2 The labeling problem . . . 16
- 3.3 Markov random fields (MRF) . . . 17
  - 3.3.1 Definition of MRFs . . . 17
  - 3.3.2 Equivalence to Gibbs Random Fields . . . 18
  - 3.3.3 Definition of the energy function . . . 19
  - 3.3.4 Design of the energy function . . . 20
- 3.4 Hierarchical Markov Random Field . . . 22

4 Inference **24**
- 4.1 Estimates . . . 24
  - 4.1.1 The maximum a posteriori (MAP) estimate . . . 24
  - 4.1.2 Posterior minimum mean squares (MMSE) estimate . . . 25
  - 4.1.3 Marginal posterior mode (MPME) estimate . . . 25
  - 4.1.4 Discussion . . . 25
- 4.2 Approximate inference methods . . . 26
  - 4.2.1 Sampling (Monte Carlo) methods . . . 26
  - 4.2.2 Graph Cuts . . . 28
  - 4.2.3 Message passing methods . . . 29
- 4.3 Belief propagation . . . 30
- 4.4 Generalized belief propagation . . . 36

5 Speedup techniques **44**
5.1 Overview 44
5.2 Hierarchical initialization 46
5.2.1 Hierarchical initialization in 2D 46
5.2.2 Hierarchical initialization in 3D 49
5.3 Active-message technique 49
5.3.1 Exact variant 49
5.3.2 Approximate variant 50
5.4 Caching and Multiplication 51
5.4.1 Caching and multiplication in 2D 51
5.4.2 Caching and Multiplication in 3D 55
5.5 Accelerating MAP estimation 58

6 Experimental Results **67**
6.1 Overview 67
6.2 Image segmentation problem 67
6.3 Results 70

7 Conclusion **73**

Notation **77**

References **80**

Chapter 1

Introduction

1.1 Motivation

In many fields of application we face the problem of *segmenting* an image into semantically coherent regions. For instance, to investigate the distribution and movement of cells in bio-molecur images, it is indispensable to first locate the cells. Or, in medicine we are interested in extracting anatomical structures from CT or MRT images. These are solely two examples for the diverse range of applications that extends from biology, medicine, face recognition to the evaluation of satellite data. Usually human experts are entrusted with evaluating and segmenting the images but the ever-growing number of images compels us to develop automatic solutions. Thus, we search for efficient segmentation algorithms that deliver accurate results, even if the image exhibits complex detailed structures and is afflicted by measurement noise.

Over the past years, many researches proposed segmentation methods for various scenarios. However, only few can cope with strong measurement noise and three-dimensional image data. Also, they often operate locally which means that they do not incorporate global constraints to guide the segmentation result. But there are also promising segmentation approaches that overcome many of these pitfalls. One example are methods that rely on *probabilistic graphical models*. We do not attempt to promote them as the holy grail of image segmentation but they often perform surprisingly well on problems that shipwreck competing segmentation methods. In comparison to point-based, edge-based or region-based methods they can model *global* constraints and are less sensitive to noise.

The most common form of probabilistic graphical models are *Markov random fields* that offer two types of potentials to model the image constraints. Data potentials encode the similarity between the search segmentation image x and the observed image y, whereas smoothness potentials between neighboring pixels of x allow us to encode prior knowledge about the image. For each pixel of x we store the probabilities for all its possible labels, i.e. the semantic region to which it can pertain. Once we have modeled all potentials, we can solve our segmentation problem by *inferring* the most likely combination for x that best suits our constraints.

Unfortunately, it is an NP hard problem [4] to exhaustively explore all possible pixel combinations for x. Thus, for complex images we have to resort to *approximate inference algorithms*, such as Monte Carlo chain (MCMC) methods, graph cuts, or message passing algorithms. For a long time MCMC methods have been the common

choice for inferring on Markov random fields. They draw samples from the image for approximating the unknown probability function of $P(x|y)$. The problem is that they converge very slowly since many samples are required to obtain a good estimate. Since the invention of more efficient inference algorithms, such as graph cuts and message passing algorithms, they are no longer state-of-the-art. Whether to advocate graph cuts or message passing algorithms, is hard to say. Both algorithms are probably on a par but as the graph cuts algorithm is restricted to certain potential structures we prefer message passing algorithms. The most popular variant is the belief propagation algorithm (BP). It is very fast and performs well on many graphs. However, for graphs with many cycles it suffers from poor convergence accompanied with unpleasant accuracy. Recently, Yedidia et al. [39] proposed a derivate of this algorithm called *generalized belief propagation (GBP)* that enhances the performance and stability of BP. A problem is that the algorithm is fairly slow and therefore limited to applications with simple graphs. Also, we found only two propositions [19] [27] to accelerate the algorithm. Both techniques require a beneficial structure of the smoothness potential function for sparing redundant label configurations. But as we like to retain flexibility for the potential functions, we can hardly benefit from these techniques.

In this thesis, we present four acceleration techniques that do not restrict the potential functions and still promise significant speedup gains. We call them (1) *hierarchical initialization*, (2) *active message technique*, (3) *caching and multiplication*, and (4) *acceleration for map estimate*. The first technique is an adaption of the multigrid BP reported by Felzenswalb et al. [8]; the other three are novel to the best of our knowledge.

The techniques are experimentally evaluated on synthetic data and one real-world segmentation problem from biology.

1.2 Outline of the thesis

The next chapter gives an overview of image segmentation and discusses the ideas and properties of various segmentation methods. We shall prefer a segmentation approach that is based on probabilistic graphical models which we elucidate in chapter 3. A very promising graphical model for computer vision are Markov random fields. They support a range of efficient inference algorithms that we shall investigate in the forth chapter. Among them are the propagation (BP) algorithm and its advanced derivate, the generalized belief propagation algorithm (GBP). The GBP algorithm stands out for its accuracy and flexibility, but is computationally expensive. As we pursue to accurately solve a complex segmentation problem, we decide to focus on speedup techniques for GBP which we present in chapter 5. Their potential is indicated by experimental results in chapter 6 and in the last chapter we conclude.

Chapter 2

Image segmentation

2.1 Segmentation methods

In computer vision, an important task is to partition an image into regions that represent information of interest. Commonly referred to as *image segmentation*, this process plays an important role in many fields of applications, e.g. in medical imaging, face recognition, machine vision, or object location within satellite data. The typical approach consists of two steps.

1) We define a *homogeneity criterion* which has to be similar among elements of the same region and different among elements of adjacent regions [26]. The homogeneity criterion is motivated either directly from the data (e.g. pixel intensity, color or texture) or from prior knowledge about the image and the application.

2) Within this constrained solution space we apply a *segmentation method* that meets our requirements as accurately and efficiently as possible.

Unfortunately, many segmentation problems are mathematically ill-posed which makes it impossible to derive a unique solution. The ambiguity merely stems from two phenomena: On the one hand, images are mostly degraded by measurement noise. Photoelectric interactions and nonlinear responses to incident radiation in detectors or amplifiers [1] as well as a low resolution and poor lighting conditions may significantly distort the information content of the data. On the other hand, it is hard to appoint a unique solution because optimality succumbs our subjective interpretation.

Nevertheless, a variety of segmentation techniques has been introduced over the past years, which we may categorize as *point-based*, *edge-based*, *region-based*, and *model-based* methods. Note that this classification does not lay claim to be complete, but rather reflects a selection of popular approaches. We shall discuss them briefly to point out alternatives and motivate our final algorithm choice. For a concise overview the reader may read [36].

2.1.1 Point-based methods

As objects are often characterized by a constant reflectivity of their surface, a widely used technique are threshold methods that distinguish objects from their background.

They associate each pixel with a segment if its value exceeds the threshold. Generally, methods based on thresholds are fast and simple, but also sensitive to intensity variation in the background and low signal to noise ratios. To some extent we can overcome these shortcomings by introducing a spatially varying threshold that increases the performance on non-uniform backgrounds. The idea is to introduce a second threshold that crops noise with high intensity values. We commonly refer to this approach as *hysteresis thresholding*.

Related to the thresholding technique are methods which rely on *classifiers*, such as *k-nearest neighbor*, *maximum likelihood* or *support vector machines*. Their strength is that they also work on multi-channel data. However, we cannot use them for spatial modeling and we have to train them manually.

In contrast unsupervised learning algorithms, also known as *clustering* methods, do not have to be trained manually. As an example, let us inspect the idea of *fuzzy connectedness* methods comprising three fundamental steps: First, they define a fuzzy affinity among all image points. Second, they compute the fuzzy connectedness as the strongest path with respect to the weakest affinity link between any two points. Finally, segmentation boils down to aggregating all pixels into the same segment that exceed a preset threshold. But also fuzzy connectedness, as well as other clustering techniques, such as *k-means* or *expectation-minimization*, are not the silver bullet because they still show great deficiencies in modeling spatial structures.

2.1.2 Edge-based methods

Thus far we have experienced segmentation methods that unite pixels to regions based on their similarity. An alternative is to solve the dual problem. We can specify the contour or boundary of a region by locating pixel discontinuities, i.e. neighboring pixels of differing intensity. The idea is to first employ an edge detecting operator to obtain a coarse guess for the segmentation result. Afterwards we can refine the result by combining edges to chains that hopefully correspond to the contours of the regions. Within the last decades various edge-based segmentation algorithms have been devised, mainly differing in the way they combine edges and the amount of prior knowledge they require.

A very simple approach for instance are methods based on *edge image thresholding*. As the name might indicate, these algorithms neglect all pixel discontinuities that fall below a certain threshold. Yet the drawbacks of this simple approach are the same as for thresholding methods: the choice of the optimal threshold and the sensitivity to image noise and intensity variations. More sophisticated algorithms are *edge relaxation methods* that improve the segmentation quality by correlating the confidence in an edge to the support of neighboring edges. Another approach are *border tracing methods* proceeding in two steps: First, they try to locate a likely border point of the image, and then they trace the border by recursively searching for the most probable border point in the neighborhood. Unfortunately, all of these edge-based methods have in common that they are heavily affected by poor edge detection results due to measurement noise in the image data. A more detailed overview of the presented methods can be found in [16].

If we know the shape and size of the object, we can simplify the segmentation problem to an object search within the image. One prominent candidate for detecting

simple shapes, such as lines, circles, and ellipses, is the *Hough transform*. The basic idea shall be exemplified by the *linear Hough transform* for line location. We can express all possible lines going through an image point with coordinates (x_0, y_0) by the parametric equation $r(\theta) = x_0 \cos\theta + y_0 \sin\theta$. Transformed into parameter (Hough) space (r, θ) we get a sinusoidal curve that represents all lines going through image space coordinates (x_0, y_0). Hence, if we superimpose the curves associated with two points, the intersecting point of the curves marks the parameters of the image line going through both points. Similar to the linear version of the Hough transform, we can apply the same principle to other shapes described in analytic equations. There is even a modification to the algorithm called the *Generalized Hough transform* that extends the algorithm to arbitrary objects that we cannot describe analytically. It aims to correlate locations and orientations of potential image features to parameters in the Hough space. When the image is denoised in a preprocessing stage, the Hough transform is an efficient and relatively robust method to detect parameterized objects. However, a severe limitation of the algorithm are the memory and computation cost for objects with several parameters. In practice we are therefore confined to simple shapes such as lines, circles and ellipses. For further information the interested reader is referred to [13].

2.1.3 Region-based methods

Opposed to edge-based methods the class of *region-based* methods exploit the similarities of pixels to construct segmentation regions of maximum homogeneity. The criterion for homogeneity can be derived from the image data, such as the intensity value, the color, the texture, the shape or from the model by using semantic information. *Region growing* methods for instance start at an image point (seed point) and subsequently merge neighboring points with its region provided that they meet the homogeneity criterion. A popular representative of *region growing methods* is the *splitting and merging* method. Inhomogeneous image regions are initially split into subregions and then merged to bigger regions as long as they accord with the homogeneity criterion.

An alternative to region-growing methods are *watershed* methods. If we relate the intensity values of an image to altitudes, we can interpret the image as a topographic map. In this map region edges are associated with local maxima, intuitively called watersheds, whereas low-gradient region interiors correspond to catchment basins. The basic idea of watershed algorithms is to fill the catchment basins with water, thereby assigning all flooded points within a catchment basin to the same segment. Unfortunately, in noisy and highly detailed images this method often ends up in oversegmentation with possibly thousands of catchment basins. To alleviate this undesired behavior we can fall back on *seeded watershed* methods where we exclusively flood previously marked catchment basins. But which regions do we mark beforehand and when do we merge regions? Hitherto these questions have not been answered in general.

2.1.4 Model-based methods

The probably most powerful, yet intricate methods for image segmentation rely on *deformable models* or *probabilistic graphical models*. The philosophy behind *de-*

formable models is to create dynamic models whose shape evolves under the influence of internal and external forces until it hopefully fits to the desired segment. More formally, we search for a contour C that minimizes the energy function $E(C) = \lambda E_{\text{data}} + (1-\lambda) E_{\text{prior}}$, where E_{data} denotes external forces that are computed from the image data to drive the curve or surface towards the desired position, and where E_{prior} stands for internal forces or prior knowledge that is computed to smooth the shape.

The general representation of the contour is either *polygonal*, *parametric* or *implicit*. The *polygonal* form is the simplest but requires many points to model the curve. In *parametrized* form, we typically adopt to the edges of an image by defining a curve or "snake" $c(s) = (x(s), y(s))$, where the parameter s varies over a certain interval, e.g. [0,1]. Some problems of snakes are their inability to nestle against concavities within object boundaries and their difficulties in locating borders when initialized too far away from them. Also, snakes are per definitionem closed and not allowed to cross [36]. A remedy against this problem are level sets, an *implicit* representation. Level sets are iso-surfaces of a function ϕ, i.e. a surface that is defined by $\phi(x) = c$ for some constant c. Remarkably, the function ϕ also implicitly defines a hyper-surface[1] S which encloses an image region. More formally, we can write $S := \partial R$, where $R := \{\mathbf{x} | \phi(\mathbf{x}) < 0\}$ and ∂ denoting the boundary of region R. When we search for a segmentation, formalized as S, rather than modifying S directly as for the case of parametric representations, we alternate our function ϕ which implicity affects S. With level sets we can define partial differential equations that describe the change of ϕ during iterations and solve them with finite difference methods. The merits of this approach are that it is straightforward to implement and can handle topology changes, intersections and superpositions. But they are expensive both in computation time and memory. Further information on this topic can be found under [36] and the study of [22] who compare parametric and implicit methods for deformable models.

Segmentation methods based on *probabilistic graphical models* strike a different path. They treat image points as random variables of segment labels and infer their probability values according to Bayesian decision theory. The fundamental approach of Bayesian decision theory is to quantify the costs of wrong and correct segmentation decisions in an attempt to minimize the expected loss of our final decision. An intuitive way of representing and visualizing the obtained joint probability distributions are graphical models, among which *Markov random fields* are probably the most prominent. They clarify dependencies and conditional independence among random variables and support efficient inference and learning algorithms. In other words, graphical models allow us to answer the following two questions: What is the most likely segmentation setting of the underlying random variables? And what are the marginal posterior probabilities over the random variables? The former question leads us to the maximum a posteriori (MAP) estimate, whereas the later drives us to the minimum mean-square error (MMSE) estimate. At this point we can recognize an interesting link between methods based on probabilistic graphical models and those depending on deformable models. In Markov random fields for instance, we can define node potentials that correlate to the probabilities of the random variables and edge

[1] a surface in Euclidean space with dimension greater than three

potentials that correlate to prior knowledge we like to incorporate. If we associate the potentials with energies, the minimization of the total energy according to the MAP estimate bears astonishing similarity to the energy minimization problem for deformable models.

2.1.5 Discussion

Particularly for noisy and highly detailed data, both model-based approaches seem to deliver more plausible segmentation results than thresholding, edge-based or region-based methods. Deformable models are notorious for requiring both high computation and memory cost, whereas approaches integrating probabilistic graphical models require some hyperparameters to be set for modeling the potential functions. These can be either set manually or learned by algorithms.

To make this clear we do not regard one segmentation method as the ultimate solution for arbitrary segmentation tasks. Nonetheless, in this thesis we like to concentrate on segmentation methods that are potentially accurate for large biological and medical images degraded by measurement noise. Comparing the presented segmentation approaches, we decide to focus on probabilistic graphical models, as the literature offers many efficient and well understood algorithms for inferring an approximate, sometimes even exact solution to a non-convex optimization problem. In contrast to methods using deformable models, we expect them to minimize the energy equation problem similarly well but cheaper with respect to computation and memory cost.

Chapter 3

Probabilistic graphical models

Probabilistic graphical models are a widespread technique for both understanding and formalizing probability distributions over random variables. The structure of graphical models tells us how the random variables represented by nodes depend on each other via edges. The weight of edges typically captures the probability of a set of variables. Thus, graphical models permit us to encode qualitative as well as quantitative relations among random variables in a compact form. From the viewpoint of image segmentation, two other aspects of probabilistic graphical models play a vital role.

1) *Conditional independence*: We can observe conditional independence relationships between variables directly from the graphical model without evaluating probability distributions. Recall that a hidden variable X is *conditionally independent* of variable Y given an observed variable V if and only if

$$P(X|Y,V) = P(X|V).$$

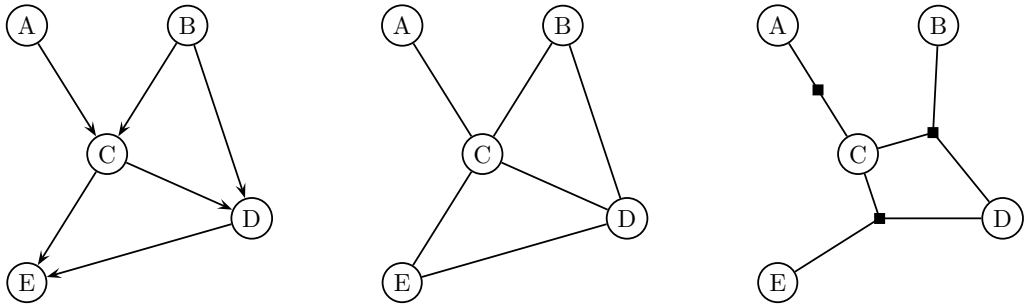
If $p(X,V) > 0$, we can also say that these variables are conditionally independent if they satisfy

$$P(X,Y|V) = P(X|V)P(Y|V).$$

The shorthand notation of conditional independence is $X \perp\!\!\!\perp Y|V$. The ability of graphical models to encode this property is important because we can simplify the graph structure and accelerate inference and learning algorithms working on this model.

2) *Efficient inference algorithms:* Graphical models support efficient algorithms for inferring joint and marginal probability distributions, such as graph cuts and message-passing algorithms.

In general, we differentiate three main kinds of graphical models: *directed acyclic graphical models*, *undirected graphical models* and *factor graphs*. In the following we review their basic ideas in order to compare their suitability for image segmentation problems. A nice overview of probabilistic graphical models can be found under [11].



(a) Directed graph (b) Undirected graph (c) Factor graph

Figure 3.1: Three kinds of graphical models (from [11])

3.1 Graphical Models

3.1.1 Directed acyclic graphical models

In a directed acyclic graphical model (DAG model) all edges are directed and form a graph without cycles. Figure 3.1(a) depicts a small example graph which corresponds to the factorization of the joint probability distribution:

$$P(A, B, C, D, E) = P(A)P(B)P(C|A, B)P(D|B, C)P(E|C, D)$$

Generally, the DAG model *factorizes* a probability distribution into a subset of nodes according to:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|X_{\mathrm{pa}(i)})$$

where $\mathrm{pa}(i)$ denotes the parents of node i, i.e. the set of nodes that have an outgoing edge to i.

This means that DAG models allow us to express the joint probability distribution as a product of conditional probability distributions. Since it might be obscure how conditional independence carries over to DAG models, we shortly restate the definition.

In a DAG model, we say that X and Y are *conditionally independent* of $\mathcal{V}$ if $\mathcal{V}$ *d-separates* (dependency separates) X from Y, i.e. if every undirected path between X and Y is blocked by $\mathcal{V}$. We say that a path is *blocked* by $\mathcal{V}$ if there is a node W on the path such that either

1) W has converging arrows along the path ($\rightarrow W \leftarrow$) and neither W nor its descendants are observed (in $\mathcal{V}$), or

2) W does not have converging arrows along the path ($\rightarrow W \rightarrow$ or $\leftarrow W \rightarrow$ or $\leftarrow W \leftarrow$) and W is observed, i.e. $W \in \mathcal{V}$.

In Figure 3.1(a) for instance $A \perp\!\!\!\perp B$ is true since all paths are blocked. However, $(A \perp\!\!\!\perp B|C)$ is false since $A \rightarrow C \leftarrow$ is not blocked [11].

3.1.2 Undirected graphical models

Undirected graphical models contain as the name suggests undirected edges and allow cycles in contrast to DAG models. A joint probability distribution can also be written as a factorization over subsets of variables:

$$P(x) = \frac{1}{Z} \prod_j \psi_{C_j}(x_{C_j})$$

where $x = (x_1, \ldots, x_S)$, and

$$C_j \subseteq \{1, \ldots, S\}$$

are subsets of the set of all variables, and $x_{C_j} \equiv (x_i | i \in C_j)$. The subsets C_j are also called *cliques*, i.e. any two elements of C_j are neighbors of each other. Without loosing generality we typically assume that C_j are maximal cliques to gain a unique factorization.

Associated with each clique C_j is a *potential function* $\psi(x_{C_j})$. As these functions can be arbitrary as long as they are nonnegative, we need a normalization constant Z to ensure that $P(x)$ integrates to 1. In the context of Gibbs distributions, Z is commonly known as the *partition function*. It equals the sum (or integral for the continuous case) of the joint probability distributions $P(x)$ pertaining to all possible settings of x. More formally, we define it by:

$$Z = \sum_x \prod_{C_j} \psi_{C_j}(x_{C_j})$$

Hence, for the example graph of figure 3.1(b) we can state the joint probability distribution as:

$$P(A, B, C, D, E) = \frac{1}{Z} \psi_1(A, C) \psi_2(B, C, D) \psi_3(C, D, E).$$

The *conditional independence* relationship in undirected graphical models is straightforward. We say that X is conditionally independent of Y given $\mathcal{V}$ if every path between X and Y contains some node $V \in \mathcal{V}$. Consequently, a variable X is conditionally independent of some Y given its neighbors:

$$X \perp\!\!\!\perp Y | \mathrm{ne}(X), \qquad \forall Y \notin \{X \cup \mathrm{ne}(X)\}.$$

In our example E is conditionally independent of B if we observe C and D.

3.1.3 Factor graphs

Factor graphs represent bipartite graphs with the following two types of nodes: *Variable nodes* x with the usual connotation and *factor nodes* $f(\cdot)$ corresponding to potential functions in undirected graphical models. In figure 3.2 we see a small factor graph example. The circles in the graph label variable nodes, whereas filled dots stand for factor nodes. The factor nodes depend on all connected variable nodes and are suitable to compose the joint probability distribution. For the example we obtain:

$$P(A, B, C, D, E) = \frac{1}{Z} f_1(A, C) f_2(B, C, D) f_3(C, D, E)$$

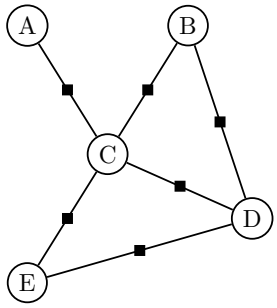


Figure 3.2: Factor graph with pairwise potentials (from [11])

Figure 3.3: Problems with undirected graphs and factor graphs (from [11])

The corresponding general equation is

$$P(x) = \frac{1}{Z} \prod_j f_{F_j}(x_{F_j})$$

where F_j is defined analogous to C_j for undirected graphical models.

The same applies to *conditional independence*. In factor graphs, we say that X and Y are conditionally independent of $\mathcal{V}$ if every path between X and Y contains some node $V \in \mathcal{V}$. A *path* is a sequence of neighboring nodes, i.e. nodes that share a common factor. One might be tempted to consider factor graphs as an alternative notation for undirected graphical models, but there are some significant differences:

1) *Expressiveness*: Factor graphs are more expressive than undirected graphical models since they can depict a more constrained subspace of probability distributions. In figure 3.1(b) we see an undirected graphical model and in 3.1(c) and 3.2 two factor graphs. They all share the same neighbors and therefore represent the same conditional independence relationship. Nonetheless, figure 3.2 differs from 3.1(c) in stating that all factors are pairwise functions. The undirected graph in 3.1(b) cannot model these subtleties.

2) *Unique representation*: Factor graphs are also advantageous because they disambiguate the representation for a probability distribution. In contrast to undirected graphical models, factor graphs exhibit a unique representation. We no longer have to agree on exclusively considering maximal cliques.

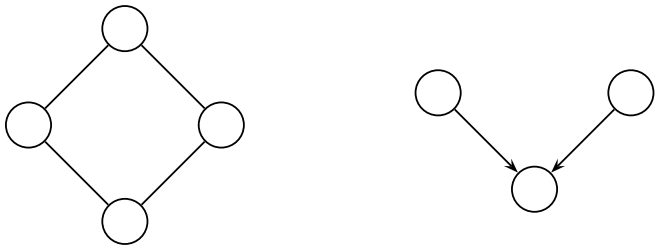


Figure 3.4: Expressive power of directed and undirected graphs (from [11])

Directed graphical models show their strength when it comes to explicating causal generative models. In figure 3.3 for instance, two nodes R and S are *marginally independent*, i.e. conditionally independent on the empty set, but conditionally dependent given G. Neither undirected graphical models nor factor graphs manage to represent these dependency relationships. Another helpful property of DAG models is that they factorize joint probability distributions by conditional probability distributions. In other words, we do not have to bother with computing the normalization constant Z as in undirected graphical models and factor graphs.

However, also directed graphical models have significant deficiencies. As figure 3.4 highlights, the expressive power of directed graphical models is limited. No matter how we direct the edge in the graph, two non-adjacent parents will share a common child. This means that under the definition of conditional independence these nodes depend on each other in DAG models, although they are independent in undirected graphs. Also, we can always construct an undirected graphical model out of a DAG model that may drop some independence relationships but covers the same probability distribution. We only have to connect all the parents of a common child and convert directed into undirected edges. We call this process *moralization*.

The decisive argument against DAG models, however, is the requirement to be acyclic. For many segmentation problems this is too restrictive, since we typically model images as grid graphs where each pixel or voxel corresponds to a random variable and is connected to its four neighbors in 2D, respectively six neighbors in 3D.

While both factor graphs and undirected graphical models seem to be legitimate choices for modeling images in segmentation problems, we merely stick to *Markov random fields*, a special subclass of undirected graphical models. Markov random fields impose further constraints on the probability function that are important for efficient inference algorithms, such as graph cuts or message-passing methods. They shall be introduced shortly but let us first get a notion of the *labeling problem*, a generalization of the image segmentation problem.

3.2 The labeling problem

As the image segmentation problem can be regarded as a special case of the *labeling problem*, we like to present its general notation. It is commonly used in many com-

puter vision problems and rests on a series of basic definitions that we have to clarify before we can state the labeling problem. We present the concepts *site*, *image*, *image feature*, *configuration* and *label*.

Similar to [35] and [32], let

$$S = \{1, \ldots, n\}$$

be a set of n sites, where a *site* s represents an element of a pixel (voxel) grid. Then we define an *image*

$$x = (g, l)$$

on S which is nothing else but an array of *image features*, e.g. intensities, edges, boundaries, or labels. In this example, g corresponds to the *pattern of intensities*, and l to a *label configuration*. Note that a pattern or synonymously a *configuration* $x = (x_s)_{s \in S}$ refers to a collection of quantities x_s. Consequently, the set of all patterns $g = (g_s)_{s \in S}$ of intensities $g_s \in \mathcal{G}$ is $\mathbf{G} = \mathcal{G}^S$. A label configuration or labeling is written as $l = (l_u)_{u \in U}$ with labels l_u from a set

$$\mathcal{L} = \{l_1, \ldots, l_k\}$$

of k labels and the space of label configurations is $\mathbf{L} = \mathcal{L}^U$. With these definitions we can denote the set of all images x by $\mathbf{X} = \mathbf{G} \times \mathbf{L}$, or alternatively as $\mathbf{X} = \prod_{s \in S} \mathbf{X}_s$ where $\mathbf{X_s}$ is the finite space of states x_s.

Let us assume that image $x = (g, l)$ is generally *hidden* to the observer. If we only observe the pattern of intensities g correctly, we take the *observed* image

$$y = g$$

as a starting point to infer x. In other words, we face a *labeling problem*, since we search for the proper label from the label set $\mathcal{L}$ of each site s based on the image features of y.

Note that the labeling problem merely appears in the disguise of queries like $P(X|Y = y)$, where we ask for the quantities of the random variable X given the observed image value y for variable Y. We prefer this notation since it is more compact than specifying all image features. We will, however, convert to explicitly naming the image features when we feel it is appropriate.

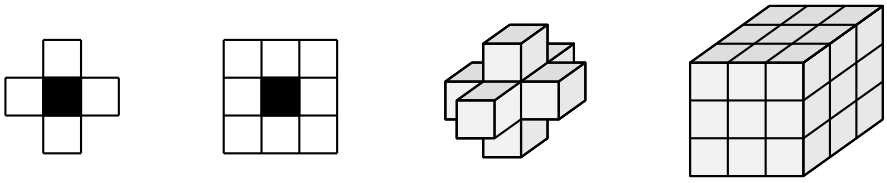
3.3 Markov random fields (MRF)

3.3.1 Definition of MRFs

A *Markov random field* (MRF), also known as *Markov network*, is defined as a random vector $X = (X_s)_{s \in S}$ on the probability space $(\mathbf{X}, P)$ with respect to a neighborhood system ∂, that fulfills two requirements for all $x \in X$:

1) *Positivity*: The probability distribution P on $\mathbf{X}$ has to be strictly positive:

$$P(x) > 0$$



(a) 4 neighbors (b) 8 neighbors (c) 6 neighbors (d) 26 neighbors

Figure 3.5: Typical neighborhood systems for a Markov random field. (a) and (b) show first- and second-order neighborhood system in 2D. (b) and (c) depict the same for 3D.

2) *Local Markov property*:

$$P(X_s = x_s | X_t = x_t, t \neq s) = P\left(X_s = x_s | X_t = x_t, t \in \partial\{s\}\right)$$

where the *neighborhood system* ∂ is a collection

$$\partial = \{\partial\{s\} : s \in S\}$$

of sets with the conditions

1) $s \notin \partial\{s\}$

2) $s \in \partial\{t\}$ if and only if $t \in \partial\{s\}$

Based on the neighborhood definition, we call sites $t \in \partial\{s\}$ *neighbors* of s, which we may often denote as $s \sim t$.

The second property of the definition for MRFs states that the image features of a site s only depend on the image features of its neighboring sites. Thus, it vastly accelerates the examination of conditional independence relationships. In figure 3.5 we see a selection of widespread neighborhood systems in computer vision.

3.3.2 Equivalence to Gibbs Random Fields

Markov random fields can either be expressed by their joint probability distribution or according to the local Markov property by local conditional distributions. In practice, we stick to the former possibility since the *Hammersley-Clifford theorem* paves the way for specifying the joint probability distribution by a Gibbs distribution. More precisely, the theorem shows the equivalence of Markov random fields and Gibbs random fields.

We formally define a *Gibbs random field* as a random vector $X = (X_s)_{s \in S}$ on the probability space $(\mathbf{X}, P)$ that fulfills two requirements for all $x \in X$

1) *Positivity*: The probability distribution P on $\mathbf{X}$ has to be strictly positive:

$$P(x) > 0$$

2) *Gibbs distribution*: Its complies to the Gibbs (Boltzmann) distribution:

$$P(x) = \frac{1}{Z} e^{-\beta E(x)}$$

where Z denotes the *partition function* used to normalize $P(x)$

$$Z = \sum_{x} e^{-\beta E(x)}$$

and

$$\beta = \frac{1}{kT}.$$

T is the *temperature*, commonly assumed to be 1, k is the *Boltzmann's constant* and $E(x)$ denotes an *energy function*.

If we restrict the potential functions $\psi_C(x_C)$ of a clique C to be strictly positive, we can define them by an exponential term of the Gibbs distribution:

$$\psi_C\left(x_C\right) = \exp\left(-\beta E\left(x_C\right)\right)$$

3.3.3 Definition of the energy function

The Gibbs distribution is primarily influenced by the *energy function* $E(x)$ which in turn can be expanded to a sum of *potentials*

$$E_A(x) = \sum_{A \subset S} U_A(x).$$

We say that a family $\{U_A : A \subset S\}$ of functions on $\mathbf{X}$ is a *potential* if and only if it satisfies

1) $U_\emptyset = 0$

2) $U_A(x) = U_A(y)$ if $x_s = y_s$ for each $s \in A$.

In the context of image segmentation, the subset A is equivalent to cliques C_j as proposed for undirected graphical models. If we restrict our attention to cliques of size two, we obtain an important specialization of the *energy function*

$$E(x) = \sum_{s \sim t} U_{st}(x_s, x_t).$$

Let us also assume that the potentials can be parameterized in *linear* form:

$$U_C^w = -\langle w, V_c \rangle$$

where parameter w signifies a weight function for the potentials. Then our energy function transforms to

$$E(x) = -\sum_{s \sim t} w_{st} V_{st}(x_s, x_t).$$

Transferred to our labeling problem, we may express the energy function of the *joint probability function* $P(x, y)$ as

$$E(x, y) = -\sum_{s} w_s V_s(x_s, y_s) - \sum_{s \sim t} w_{st} V_{st}(x_s, x_t).$$

The first term may be interpreted as the *data energy* E_{data}

$$E_{\text{data}}(x, y) = \sum_s U_s^w(x_s, y_s) = -\sum_s w_s V_s(x_s, y_s)$$

which assigns costs relative to the deviation from the image data. The term is for instance reasonable when we like to denoise an image prior to the image segmentation. We can model this task with a Markov random field by interpreting the label, the quantity to be estimated, as the hidden pattern of intensities that is constrained by the data energy to stay as close as possible to the observed pattern of intensities.

The second term may be regarded as the *prior energy* E_{prior} defined by

$$E_{\text{prior}}(x) = \sum_{s \sim t} U_{st}^w(x_s, x_t) = -\sum_{s \sim t} w_{st} V_{st}(x_s, x_t).$$

As it is independent of the observed image y, it is suited to encode constraints that we have prior to the recorded data. For many vision problems, we will model the prior energy as a *smoothness* term that rewards smoothness and penalizes discontinuities among neighboring labels. We therefore prefer the term *smoothness energy* and shall use it throughout this thesis. Its notation is E_{smooth} [35] [32].

Combining both energy terms with the energy function of our labeling problem, we gain an intuitive form that shall encounter us throughout this thesis:

$$E(x, y) = E_{\text{data}}(x, y) + E_{\text{smooth}}(x)$$

3.3.4 Design of the energy function

We only obtain satisfactory solutions to vision problems if we carefully design an energy function that reflects the constraints of the problem. An improper energy function will almost certainly fail in delivering accurate results, regardless of which minimization technique we apply to the energy equation.

For both the *data energy* and the *smoothness energy* we often choose among the same popular candidates: (1) the *everywhere smooth prior*, (2) the *piecewise constant prior* and (3) the *piecewise smooth prior*. They shall be briefly reviewed for the *smoothness potential function* since they give us a platform for developing more specialized energy functions.

Everywhere smooth prior

The *everywhere smooth prior* [32] rewards label configurations that are smooth everywhere. In other words, it heavily penalizes discontinuities of an image. Unfortunately, the optimal solution of the prior tends to oversmooth discontinuities. An example for this prior is a non-decreasing function depending on the label difference between two neighboring sites. We denote it by

$$U_{st}^w(x_s, x_t) = -w_{st} V_{st}(|x_s - x_t|) = w_{st}|x_s - x_t|.$$

An example graph for the potential function U_{st} is depicted in figure 3.6. We typically use this metric for vision problems like *stereo matching* and *image denoising* [28].

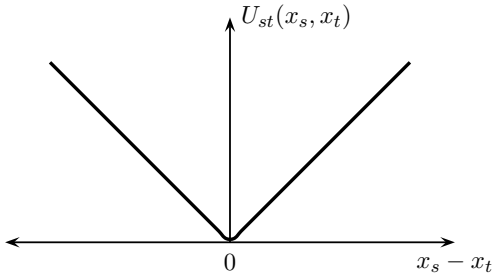
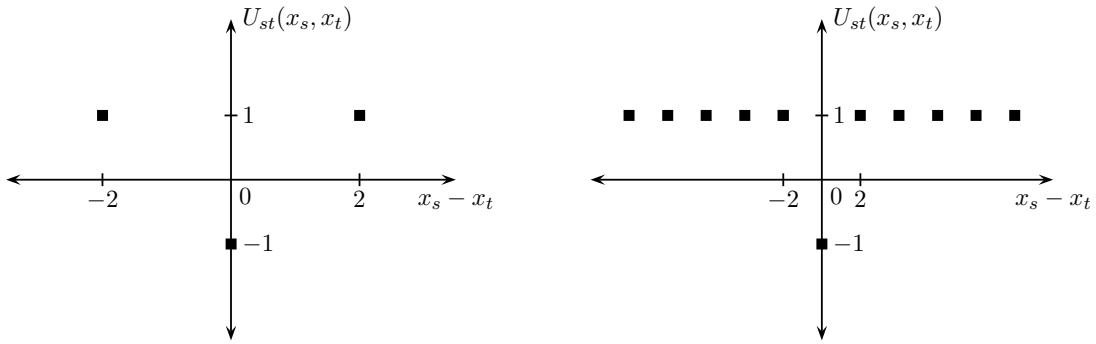


Figure 3.6: Graph of $U_{st}(x_s, x_t)$ for a everywhere smooth prior



(a) Ising model

(b) Potts model

Figure 3.7: Graphs of $U_{st}(x_s, x_t)$ for two piecewise constant priors.

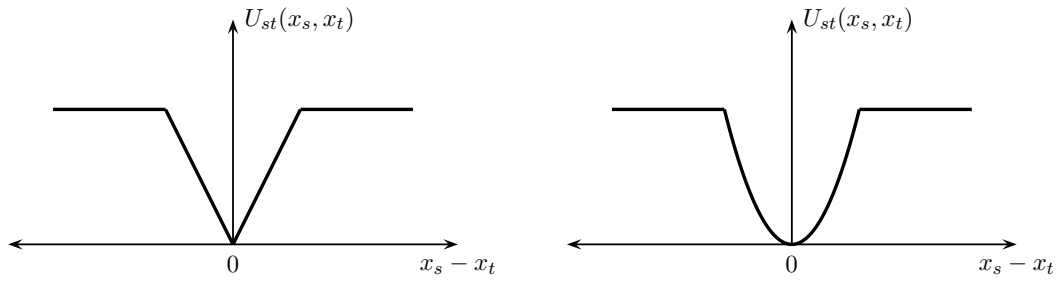
Piecewise constant prior

The *piecewise constant prior* [32] rewards label configurations that consist of areas with constant labels. It represents a very simple prior that is theoretically able to consider discontinuities for its optimal solution. For many vision problems, however, this prior is inept of encoding all needed constraint between neighboring labels. Two examples that enjoy great popularity in computer vision are the *Ising model* for binary segmentation problems and its generalization to more labels, the *Potts model*.

The *Ising model* gained its merits in statistical physics for studying phase transitions of magnetic systems. It assumes that each element is magnetically coupled to its neighbors and can either take the value 1 if it is ferromagnetic or -1 if it is antiferromagnetic. Remarkably, this model can be translated to many other systems that share the same dimensions and the same symmetries [23]. For this reason, Ising models are attractive for explaining the behaviour of interacting sites in binary labeling problems. We can formalize the Ising model as

$$V_{st}(x_s, x_t) = x_s x_t$$

where $x_s \in \{-1, 1\}$ and $x_t \in \{-1, 1\}$. In figure 3.7(a) we see an example graph for an Ising model.



(a) Truncated linear model (b) Truncated quadratic model

Figure 3.8: Graphs of $U_{st}(x_s, x_t)$ for two piecewise smooth priors

The *Potts model* generalizes the Ising model to labeling problems with more than two labels and can be formalized as

$$V_{st}(x_s, x_t) = 2\delta(x_s, x_t) - 1$$

where δ is the Kronecker delta[1]. Figure 3.7(b) shows the graph of the potential function $U_{st}(x_s, x_t)$ for the Potts model.

Piecewise smooth prior

The *piecewise smooth prior* [32] rewards label configurations that comprise pieces of smoothly varying labels. The basic idea is to set an upper bound for the penalty of the everywhere smooth prior which allows us to obtain smoothed regions as well as discontinuities in the optimal solution. We can formalize the prior as

$$U^w_{st}(x_s, x_t) = \min(|x_s - x_t|^k, U_{\max})$$

with $k \in \{1, 2\}$ and $U_{\max}$ as the upper bound [28]. In figure 3.8(a) we see the potential graph for $k = 1$, yielding the *truncated linear model*. For $k = 2$ we obtain the *truncated quadratic model* which is depicted in figure 3.8(b).

3.4 Hierarchical Markov Random Field

For highly complex Markov random fields it becomes intractable to minimize the corresponding energy. A natural resort is to solve the problem by a sequence of reduced versions of the Markov random field. We start by determining the estimates on the coarsest version and use it to guide the inference on the next finer image. We commonly refer to such models as *hierarchical Markov random fields (HRMF)*. A hierarchical Markov random field comprises two parts: a *pyramid* of interdependent Markov random fields and an *observation field*. Each hierarchy level is associated with a simplification (often a scaled version) of the original Markov random field. In the context of HRMFs we call the original Markov field observation field [17] [12].

[1] $\delta(x_s, x_t)$ takes the value 1 when $x_s = x_t$ and 0 when $x_s \neq x_t$.

HMRFs are constructed in a bottom-up approach. Each hierarchy node depends on a specific block of the observation field insofar as its label and partition functions are determined by the nodes of the block. Once we have initialized all nodes, we can apply the inference algorithm from top to bottom. For each hierarchy level we solve the energy minimization problem and project the result on the next finer level where it can be refined with the help of additional image information. At the bottom of the pyramid we obtain an approximate solution to the observation field. Again, we incorporate missed image details and conclude with an approximation of the searched estimates.

According to [25] we can differentiate two types of HRMFs: *algorithm-based* and *model-based* [25]. The former refers to models the aforementioned models. That is, coarser levels project their results to the next finer level, where we devise a more refined version. A typical example is a *multiscale* Markov random field, which constructs a hierarchy of of differently scaled images. On the contrary, *model-based* HRMFs alternate the node and edge structure of the model to include global hierarchical information during the inference algorithm. Thus, we change the neighborhood system and the partition function of a node.

Several authors share the opinion that HMRFs often lead to much faster convergence and improved accuracy. For example, deterministic relaxation algorithms, such as ICM, feature more precise results on coarser hierarchy levels, because we face less local minima to get stuck in. Even stochastic relaxation algorithms like simulated annealing may perform better [17]. HMRFs may also simplify the modeling of prior knowledge that does not directly affect the interaction between adjacent pixels. Imagine we like to constrain the minimum size for a segmentation region. In flat Markov random fields we would have to define large neighborhood systems which is computationally intractable. With hierachy levels, though, we have a natural and efficient representation for describing image variations of large scale [2].

Unfortunately, it is not obvious how to design and adapt a HRMF for a given labeling problem. Of course, we can choose among conventional neighborhood systems, such as the quadtree model or an augmented pyramidal graph structure as in [2]. However, their structure also exhibits some major disadvantages. In quadtree models for instance adjacent pixels may be associated with different nodes on a coarser level. In augmented models which incorporate additional potentials to strengthen the affinity among adjacent nodes we may no longer factorize the joint probability function. A second problem is that we have to model meaningful transition functions within and between the hierarchy levels. For complex segmentation problems it is already a delicate matter to design suitable potential functions for the observation field.

In this thesis, we employ an algorithm-based approach in section 5.2 but solely for initializing our observation field. The advantage is that we benefit from knowledge about spatially distanced nodes while retaining our original labeling problem.

Chapter 4

Inference

In the previous section we argued that Markov random fields are a suitable representation for the labeling problem, as they allow us to efficiently infer the hidden image x based on the observed image y. In particular, we are interested in seeking the "optimal" image $\hat{x}$ where "optimal" usually refers to one of the following three modes.

4.1 Estimates

4.1.1 The maximum a posteriori (MAP) estimate

The "optimal" estimate should comply with the observed image y and the a priori knowledge about the labeling problem. The Bayesian approach subsumes both conditions in an elegant way [35] [10]. Following Bayes rule, we may interpret the "optimal" estimate as the maximal posterior probability $P(x|y)$:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)},$$

where $P(y|x)$ expresses the *likelihood* and $P(x)$ stands for the *prior distribution* of the hidden image x.

Neglecting the constant joint probability distribution of the observed image $P(y)$, we can thus define the *MAP estimate* $\hat{x}$ by

$$\hat{x} = \arg\max_{x \in \mathbf{X}} P(x|y) = \arg\max_{x \in \mathbf{X}} P(y|x)P(x).$$

An important tie can be bound to the energy function of the *conditional probability* $P(x|y)$, respectively the *joint probability distribution* $P(x, y)$. If we surmise that the measurement noise at each site s is independent, we can write

$$P(y|x) = \prod_{s \in S} P(y_s|x_s).$$

Let us also assume that the likelihood of $y = g_{\text{noised}}$ given $x = (g, l)$ obeys the Gibbs distribution:

$$P(y_s|x) = \frac{1}{Z_s} \cdot e^{-E_{\text{data}}(x, y_s)},$$

where Z_s denotes a normalizing partition function. Then maximizing $P(x|y)$ is equal to minimizing

$$E(x, y) = E(x) + E_{\text{data}}(x, y),$$

where we substituted the energy function $E(x)$ that underlies $P(x)$. As we discussed in section 3.3.3, $E(x)$ refers to the *smoothness* or *prior energy* E_{smooth} [32][35].

Thus, the MAP estimate of a probability distribution minimizes the corresponding energy function.

4.1.2 Posterior minimum mean squares (MMSE) estimate

Another popular estimate is the *posterior minimum mean squares estimate*. It computes the mean of the posterior distributions and is defined by

$$\bar{x} = \sum_{x \in \mathbf{X}} x \cdot P(x|y).$$

4.1.3 Marginal posterior mode (MPME) estimate

If the context of a label is unimportant, we may compute the *marginal posterior mode estimate*. It computes for each node x_s the maximal probability distribution value separately:

$$\hat{x}_s = \arg \max_{x_s \in \mathbf{X}_s} P(x_s|y)$$

4.1.4 Discussion

It is controversially discussed which estimate to prefer. The most popular candidate is the MAP estimate which is supported by all inference methods of this section. However, as [2] argues, the MAP estimate may be too conservative. It minimizes the probability that any node will be misclassified although segmentation problems nearly always produce misclassified nodes. Also, [29] reports that the MAP estimate demonstrates clear artifacts such as stair-step effects on images, whereas the MMSE estimate convinces with much smoother results. On the contrary, the MAP estimate is legitimated by minimizing the corresponding energy function of the graphical model, whereas the MMSE estimate lacks such a theoretical justification. Even worse, not every inference algorithm may calculate the MMSE estimate. For instance graph cuts (cf. section 4.2.2) can only compute the MAP estimate [29]. The third alternative, the MPME estimate, on the other hand leads a shadow existence in computer vision since it neglects the context of a node and we do not know any efficient inference method to compute it. In the end, we follow the consideration of [35] who questions the decisive importance of the estimate choice. Instead, we may influence our inference result more effectively by amending our prior knowledge and the choice of parameters. Under this perspective we favor the MAP estimate for the rest of this thesis, as it is the standard and allows us to compare the performance of inference algorithms in the result section.

Regardless of our preferred estimate, we have to accept, however, that its computation is an NP hard problem. More formally, we need exponential time to compute the minimal tree-width triangulation of a graph [4]. Only in special subclasses we

know methods that allow us to compute the *exact* estimate in polynomial time. For instance, in trees we can apply a message passing algorithm called *belief propagation*. In most vision problems, however, we are dealing with highly cyclic graphs, forcing us to resort to *approximate* inference methods.

4.2 Approximate inference methods

In this section, we review prominent alternatives for *approximating* a hidden image x based on an observed image y. We divide the methods into three categories: (1) *sampling (Monte Carlo) methods*, (2) the *graph cuts* algorithm, and (3) *message passing methods*. As the observed image y is fixed we may simplify the notation for the joint probability distribution $P(x, y)$ by writing $P(x)$ instead. Please keep in mind that we refer to the joint probability distribution, although we use the notation for the prior distribution.

4.2.1 Sampling (Monte Carlo) methods

We face an intractable problem when evaluating the exact estimate of the posterior distribution. The major obstacle is the partition function Z which is defined as the sum over the probability distributions of all possible configurations. A popular way of avoiding the computation of Z consists in drawing random samples from the probability distribution, thereby approximating the exact distribution closer and closer. We refer to this approach as *sampling* or *Monte Carlo* methods [25].

How do we know that Monte Carlo methods are guaranteed to converge against the target distribution? In other words, why does the probability distribution P_{X_m} of our samples, i.e. a chain of random vectors $X_1, \ldots, X_m, \ldots$, tend to P_X with increasing m? The answer is that Monte Carlo methods simulate *invariant*[1] and *ergodic*[2] *Markov chains*[3], which can be proven to converge against the same invariant distribution, regardless of the initial distribution. For a formal proof of this statement, see [35]. At this point we only like to develop a notion for the technical requirements of Monte Carlo methods. We sense that we have to be cautious when designing Monte Carlo methods since the transition probability has to satisfy various mathematical properties.

Popular examples for *Markov Chain Monte Carlo* (MCMC) methods are the

[1] We say a probability distribution μ is *invariant* if it fulfills $\mu P^n = \mu$ for every $n \geq 0$. A sufficient condition for invariance is the *detailed balance equation*

$$\mu(x)P(x, y) = \mu(y)P(y, x)$$

for all $x, y \in X$.

[2] A Markov chain is called *ergodic* if it is *recurrent*, *non-null* and *aperiodic*, i.e. we can return to each state (*recurrent*), the expected time to return to each state is finite (*non-null*) and the configuration set cannot be split in subsets that would be visited in a periodic way (*aperiodic*).

[3] A *Markov chain* is formally given by an *initial probability distribution* ν and *transition probabilities* $P_1, P_2, \ldots$ on $\mathbf{X}$. Thus, we obtain the probability of states $x_0, x_1, \ldots, x_n$ at times $0, \ldots, n$, respectively, by

$$P^{(n)}((x_0, \ldots, x_n)) = \nu(x_0)P_1(x_0, x_1) \cdot \cdots \cdot P_n(x_{n-1}, x_n).$$

Metropolis algorithm, *Gibbs sampling*, and *simulated annealing*. We shall briefly review their underlying principles:

Metropolis algorithm

Again we assume that $P(x) = P^*(x)/Z$ is computationally infeasible for direct sampling. Instead we introduce an easy-to-sample *proposal distribution* Q, i.e. a distribution that satisfies

$$P(x) \leq c\,Q(x)$$

for all values of x and a known constant c.

In the Metropolis algorithm, this proposal distribution depends on the current state $x^{(t)}$ where the superscript t labels the sequence of states in the corresponding Markov chain. The algorithm steps are straightforward: We start with an arbitrary labeling $x \in \mathbf{X}$. Then for each time step t we sample a tentative state x' from the proposal distribution $Q(x'|x^{(t)})$ which is accepted with a specific probability as the new state.

In contrast to very simple Monte Carlo methods, such as *rejection sampling*, the Metropolis algorithm is suited for high-dimensional problems because our proposal distribution $Q(x)$ does not have to resemble $P(x)$. The drawback of the algorithm, however, is its long running time. Because each sample depends on the proposal distribution of its predecessor, we have to calculate many samples to simulate random walk behavior [23].

Gibbs sampling

Another widespread sampling technique for distributions over at least two dimensions is *Gibbs sampling*. We can understand it as a special case of Metropolis sampling in which we define the proposal distributions $Q(x)$ as the conditional distributions of P:

$$P(x_i|x_{-i}),$$

where x_{-i} denotes all the variables except i.

Thus, if the conditional distribution $P(x_i|x_{-i})$ is easy to sample from we typically apply this method according to the following sampling scheme:

$$x_1^{(t+1)} \sim P(x_1|x_2^{(t)}, x_3^{(t)}, \ldots, x_n^{(t)})$$

$$x_2^{(t+1)} \sim P(x_2|x_1^{(t+1)}, x_3^{(t)}, \ldots, x_n^{(t)})$$

$$\vdots$$

The properties of Gibbs sampling are two-fold: On the one hand, this method is very easy to implement which probably contributed to its popularity. On the other hand, the algorithm also needs many computation steps until it behaves like a random walk along the probability distribution which is especially unpleasant for small step sizes [23].

Simulated Annealing

The Achillis heel of the preceding sampling methods is that they are prohibitively slow when simulating random walks. One famous technique to accelerate convergence speed is *simulated annealing* [10]. The intuitive idea is to simulate the controlled cooling (annealing) of a material towards a state of minimal energy. We account for this by introducing a temperature variable T that controls the annealing process. Beginning with a high temperature value, we seek for the lowest energy state whilst gradually decreasing the temperature according to a *cooling schedule*. In the algorithm, we take an arbitrary labeling as a starting point. A new sample is then accepted if it leads us to a lower energy state. Otherwise we still accept with a certain probability that depends on the temperature. As the temperature steadily drops, the acceptance probability also decreases. Thus, the temperature progressively restricts the freedom of movement in the probability distribution.

Theoretically, simulated annealing is able to find the global minimum of an energy function. In practice, however, optimal cooling schedules converges too slowly. For this reason we are forced to design sub-optimal schedules that are no longer guaranteed to converge against the globally optimal solution but at least do not get trapped in the same local minimum [32].

In a nutshell, Monte Carlo methods are widely used to sample from joint probability distributions, mainly due to its simplicity. Unfortunately, popular representatives, such as Metropolis sampling and Gibbs sampling, suffer from slow equilibration and are very sensitive to the initial labeling. Simulated annealing attempts to compensate for these shortcoming by using the cooling schedule that regulates the selection of samples. In practice, however, only suboptimal cooling schedules are feasible that may find a solution close to the global minimum after a long running time. A drop of bitterness is certainly that we need to adjust some parameters for simulated annealing and that our approximation of the estimate is not reproducible.

4.2.2 Graph Cuts

The *graph cuts* algorithm [14] has proven to be a powerful tool for solving energy minimization problems. It allows us to efficiently determine the exact MAP estimate for binary labeling problems and results in an approximate solution for problems with more than two labels. The central idea behind the algorithm is to reformulate the segmentation problem as a min-cut problem which we can minimize in polynomial time via the max-flow algorithm [9].

Let us discuss the idea of graph cuts in more detail to better understand its properties: We initiate the algorithm by manually selecting image nodes that pertain to different labels. At the representation level, this relates to hard-wiring the manually selected nodes to some novel label nodes called *terminal nodes*. Then it is clear that a (multiway) *cut* on this model, i.e. a subset of edges that totally separates the terminals, reflects a segmentation of the image. We also know that the pairwise potentials defined on the edges correlate to the smoothness of adjacent image nodes. This means that similar image nodes are modeled with high potential values, whereas image disparities, such as segmentation boundaries, correspond to small potential values. The quintessence is that the minimal cut through the Markov random field should delineate a reasonable segmentation of the underlying image. Graph cuts

exploits this insight very efficiently by computing the minimal cut with the help of the max-flow algorithm, which is possible due to the celebrated max-flow min-cut theorem [5].

The two most popular variants of the graph cut algorithm are the *swap-move* and the *expansion move* algorithm. Both variants compute the labeling problem iteratively. The *swap move* algorithm swap two label α and β of two subsets of nodes until a swap move does not decrease the local energy level. The *expansion move* algorithm, on the contrary, attempts to increase the set of nodes that share a specific label α until an expansion move does not yield a lower energy level [28].

The complexity for the standard α-expansion variant is quadratic in the number of nodes and linear in the number of labels. If the energy functions exhibit a simple characteristic structure we may even achieve logarithmic complexity in the number of labels [21]. Or, we may exploit in the max flow algorithm that the graph sometimes barely changes in successive iteration steps [18]. Experimental results have proven that graph cuts clearly outperforms Monte Carlo methods both in speed and accuracy. This also explains its popularity for many applications in computer vision, such as image restoration, stereo and motion, image segmentation, or medical imaging. A severe shortcoming of graph cuts, however, is its restriction to specific energy functions. Boykov et al. [3] report that the expansion move algorithm requires the potential function V_{st} to be a metric, while the swap algorithm demands that V_{st} forms a semi-metric.

4.2.3 Message passing methods

A viable alternative to graph cuts are *message-passing algorithms* such as the Viterbi, Forward-Backward, or the belief propagation (BP) algorithm. In numerous applications they demonstrate their potential to perform fast approximate inference on graphical models. They all share the idea of sending messages between nodes for finding the most likely configuration of hidden nodes. While the Viterbi and the Forward-Backward algorithm are applied in the context of *hidden Markov models (HMM)*, we may employ Pearl's belief propagation algorithm for Markov random fields. Depending on the desired estimate, it comes in two flavours:

1) The *max-product* or equivalently *min-sum* algorithm for searching the MAP estimate and

2) The *sum-product* variant for computing the MMSE estimate.

On the contrary, the graph cuts algorithm can only compute the MAP estimate, as it attempts to minimize the energy function encoded in the graphical model. The belief propagation algorithm is also less restricted in the choice of the energy function. This plays a crucial role since we seek an inference algorithm that allows us to flexibly design the graphical model for optimally reflecting our labeling problem.

Both inference approaches have in common that they are deterministic, i.e. they reproduce their solutions for the same graphical model in the same starting configuration. According to [29] the performance of both algorithms are comparable, although the graph cuts algorithm constantly finds lower energy levels which manifests in smoother results. In another study of [28] the standard belief propagation algorithm

performed significantly worse than the graph cuts algorithm on many benchmarks. Similar accuracy and convergency problems are reported by [24].

To overcome these deficits they propose a message passing variant called generalized belief propagation which they introduce in [39]. It demonstrates the same flexibility as the belief propagation algorithm but is much more accurate and converges more reliably. A severe problem, though, is its execution time. In the following two sections we first introduce the standard belief propagation that is fundamental to the generalized belief propagation algorithm. Afterwards we shall analyze possible speedup techniques for the generalized belief propagation to obtain a flexible, fast and accurate inference algorithm for solving complex labeling problems.

4.3 Belief propagation

The *belief propagation* algorithm is a message passing method that iteratively computes the marginal probabilities of a graphical model. It delivers the exact solution for undirected graphical models that are trees, i.e. graphs without loops, and approximations for loopy graphical models. It is important to note that the belief propagation algorithm is not guaranteed to converge for general graphs and the precision seems to vary with the cyclicity of the graph.

The conceptual idea behind belief propagation is straightforward. In the graphical model, we iteratively send messages between nearby nodes to update beliefs. A *belief* is simply the technical term for our approximation of the marginal probabilities, whilst a *message* incorporates all the evidence to locally evaluate these beliefs. Let us formalize this idea to sharpen our understanding of the algorithm.

We decide to explain the principal steps on pairwise Markov random fields, although it is mathematically equivalent to formulate the algorithm for directed acylic graphical models, factor graphs, or undirected graphical models. Recall from section 3.3.2 that we can write the joint probability distribution as

$$P(x) = \frac{1}{Z} e^{-\beta E(x)}$$

where Z denotes the partition function

$$Z = \sum_x e^{-\beta E(x)}$$

with

$$\beta = \frac{1}{kT}.$$

T denotes the temperature, k the Boltzmann's constant and $E(x)$ the energy function, expressed for a pairwise Markov random field as

$$\begin{aligned} E(x) &= -\sum_s w_s V_s(x_s) - \sum_{s \sim t} w_{st} V_{st}(x_s, x_t) \\ &= E_{\text{data}}(x) + E_{\text{smooth}}(x) \end{aligned}$$

Without loss of generality, we may set β to 1 and write the joint probability function $P(x)$ in potential form:

$$P(x) = \frac{1}{Z} \prod_s \phi_s(x_s) \prod_{st} \psi_{st}(x_s, x_t)$$

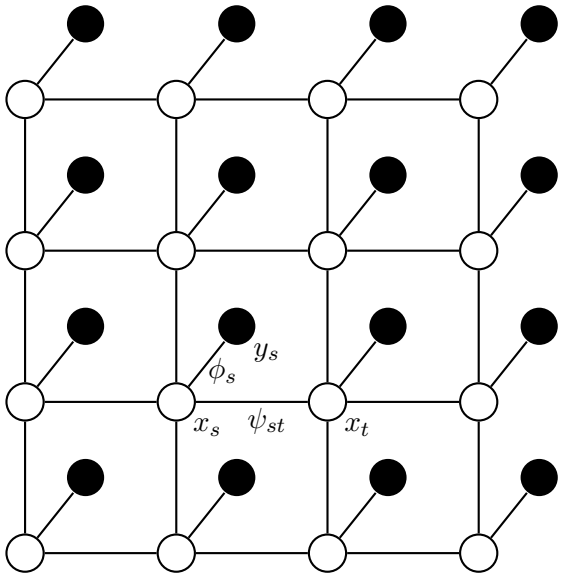


Figure 4.1: A Markov random field with pairwise potential functions.

where we define a *data potential function* $\phi_s(x_s)$ and a *smoothness potential function* $\psi_{st}(x_s, x_t)$ as

$$\phi_s(x_s) = \exp\left(w_s V_s(x_s)\right)$$
$$\psi_{st}(x_s, x_t) = \exp\left(w_{st} V_{st}(x_s, x_t)\right).$$

Keep in mind that $\phi_s(x_s)$ is a shorthand notation for $\phi_s(x_s, y_s)$. We subsume the variable y_s in the definition of $\phi_s(x_s)$, since the observed image can be regarded as fixed.

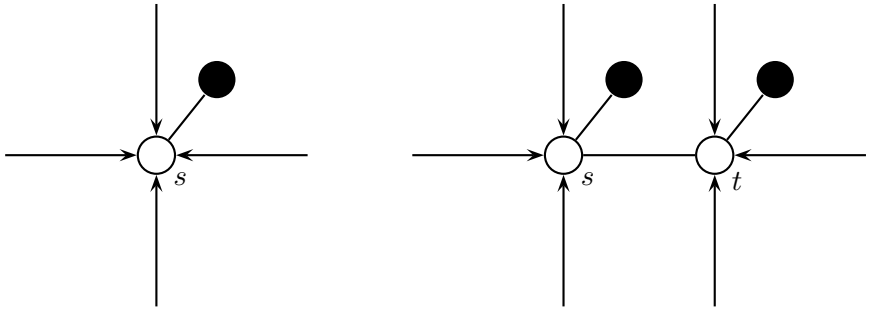
In figure 4.1 we see a square lattice pairwise Markov random field that consists of connected filled and empty circles. The filled circles denote observed image nodes y_s, the empty circles indicate hidden image nodes x_s. Connections between a filled and an empty circle correspond to data potential functions, whereas connections among filled circles are associated with smoothness potential functions.

Along each edge we can send *messages* $m_{st}(x_t)$ from node s to node t that propagate the likeliest label probabilities for t from the view of s. Formally, $m_{st}(x_t)$ is represented by a one-dimensional vector whose size equals the number of possible labels. Thus, we can derive the computation formula for the *belief* at node s as the normalized product of the data potential $\phi_s(x_s)$ and all incoming messages $m_{ts}(x_s)$.

$$b_s(x_s) = k\phi_s(x_s) \prod_{s \sim t} m_{ts}(x_s)$$

where k normalizes the sum of beliefs to 1. In figure 4.2(a) we can visually comprehend which potential function and messages determine the belief of node s.

How do we compute the message $m_{ts}(x_s)$? A common approach is to derive the



(a) Single-node belief $b_s(x_s)$ (b) Two-node belief $b_{st}(x_s, x_t)$

Figure 4.2: Diagrammatic representations of the belief formula for single-node and two-node beliefs. The arrows indicate messages that affect the corresponding hidden nodes. Undirected lines denote potential functions (from [40]).

formula through the marginalization condition

$$b_s(x_s) = \sum_{x_t} b_{st}(x_s, x_t)$$

The term $b_{st}(x_s, x_t)$ denotes a two-node belief (cf. figure 4.2(b)), which we can compose analogously to one-node beliefs.

$$b_{st}(x_s, x_t) = k\phi_s(x_s)\phi_t(x_t)\psi_{st}(x_s, x_t) \prod_{u \in \partial(s)\backslash t} m_{us}(x_s) \prod_{v \in \partial(t)\backslash s} m_{vt}(x_t)$$

where $u \in \partial(s)\backslash t$ stands for all neighbor nodes u of node s except node t. Note that incoming messages have its source outside the region of belief nodes, since we try to receive "outer knowledge" about the label probabilities of the belief nodes. For edges within the belief region we take the values of the smoothness potential function.

Combining all three formulas, we gain the *message update rule*

$$m_{ts}(x_s) = \sum_{x_t} \phi_t(x_t)\psi_{st}(x_s, x_t) \prod_{v \in \partial(t)\backslash s} m_{vt}(x_t).$$

The formula reveals the equivalence to the *sum-product algorithm* where we also summarize over products of potential functions and incoming messages. Even further, the formula exhibits the fundamental idea behind behind belief propagation. At its heart the algorithm simply changes the order of sums and products in the marginalized distribution. Instead of marginalizing over an exponential number of joint probability distributions we can as well multiply partial sums. Image that we like to compute the marginalized probability of x_1 in the directed chain graph that is depicted in figure 4.4.

$$\sum_{x_2, x_3, x_4} P(x_1, x_2, x_3, x_4) = P(x_1) \sum_{x_2} P(x_2|x_1) \sum_{x_3} P(x_3|x_2) \sum_{x_4} P(x_4|x_3).$$

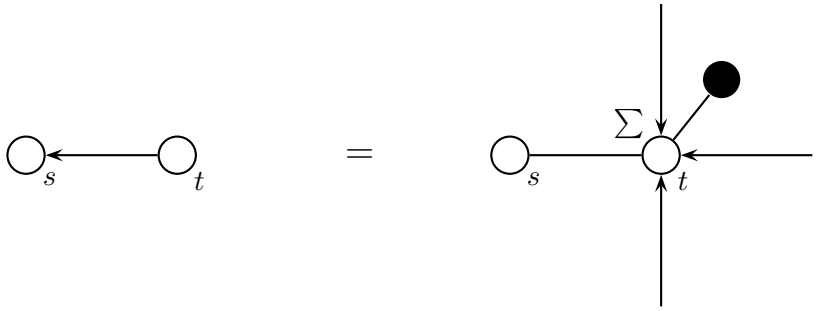


Figure 4.3: A diagrammatic representation of the message update rule $m_{ts}(x_s)$. The summatoin symbol indicates that we sum over all possible labels for node t (from [40]).

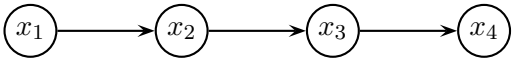


Figure 4.4: A directed chain graph for four nodes.

If we assume that each image site can choose among k labels, we notice the striking difference between both approaches. On the left hand side of the formula, we have to calculate k^n (here $n = 3$) different probability values. Yet, the reordering of sums and products on the right hand side causes only $n \cdot k$ evaluations. Belief propagation therefore reduces the computational complexity from exponential to polynomial in the number of image nodes.

The message update rule, as it is stated above, computes the minimum mean-square (MMSE) error. For obtaining the maximum a posteriori (MAP) estimate, we have to perform a minor change to the message update rule. Instead of summing over products, we have to calculate the component-wise maximum, yielding

$$m_{ts}(x_s) = \max_{x_t} \left(\phi_t(x_t)\psi_{st}(x_s, x_t) \prod_{v \in \partial(t) \setminus s} m_{vt}(x_t) \right).$$

This form is also known as the *max-product algorithm*. Alternatively, we may compute the MAP estimate with the *min-sum algorithm* that computes the minimum of the negative log probabilities:

$$m_{ts}(x_s) = \min_{x_t} \left(U_t(x_t) + U_{st}(x_s, x_t) + \sum_{v \in \partial(t) \setminus s} m_{vt}(x_t) \right).$$

The min-sum formulation directly reflects the energy function $E(x)$ and it may be numerically more stable than the product variant. Nevertheless, we will use the max-product form throughout this thesis because it did not reveal any severe numerical sensitivities in our implementation and and we are accustomed to its notation.

We have gathered all ingredients to formalize the *belief propagation* algorithm:

Algorithm 1 BP algorithm

Require: Positive thresholds ϵ and $iter_{\max}$.

for all messages m_i **do**
  $m_i^{\text{old}} \Leftarrow 1$
  for all single-node beliefs b_i **do**
    $b_i^{\text{old}} = 1$
  end for
  $d \Leftarrow 1000$ ▷ some big number
  $iter \Leftarrow 0$
  while $(d \geq \epsilon)$ and $(iter < iter_{\max})$ **do**
    for all messages m_i **do**
      Compute new message value m_i^{new}.
    end for
    $d \Leftarrow 0$
    for all single-node beliefs b_i **do**
      Compute new belief b_i^{new} with the help of messages m^{new}.
      $d \Leftarrow d + |b_i^{\text{new}} - b_i^{\text{old}}|$
      $b_i^{\text{old}} \Leftarrow b_i^{\text{new}}$
    end for
    for all messages m_i **do**
      $m_i^{\text{old}} \Leftarrow m_i^{\text{new}}$
    end for
    $iter \Leftarrow iter + 1$
  end while
  for all nodes n_i **do**
    label$(n_i) \Leftarrow \arg\max_k b_i(k)$
  end for
end for

The variable $iter_{\max}$ in line 8 lets the algorithm escape from endless loops; the variable ϵ defines a threshold for the difference of beliefs in line 15. Particularly for graphs with tight loops, we may experience that the algorithm does not find a solution because the message values circulate in loops without ever converging to a fixed point. It is still obscure under which exact circumstances the belief propagation fails to converge. We assume that the convergence behavior is probably related to the similarity to trees but how precisely is still object to research. Thus, we rely on empirical results. In [39] the authors analyzed the convergence behavior on a square lattice Ising spin glass with randomly chosen potential function values. The results indicate that the algorithm converges surprisingly often on two-dimensional images, yet rather seldom in three-dimensional images. It is questionable to generalize these results, as we cannot assume that real-world data contains purely random potential functions. For this reason we like to contribute some empirical results about the convergence behavior on biological images of two and three dimensions.

In line 10 we may ask ourselves which *update scheme* we shall apply to the message update rules. Although it barely affects the accuracy, it is an important question as it may have significant impact on the convergence speed of the algorithm. What are the common choices for the update scheme?

1) *Synchronous update scheme*: In each round we carry an old and a new value for each message. The *old* message value denotes the message value of the previous iteration; the *new* message value refers to the newly computed value. In a *synchronous update scheme*, we exclusively use old message values for the computation of new message values. Once all new message values are computed, we update all old message values with the new ones [29].

2) *Asynchronous update scheme*: In an *asynchronous update scheme* we propagate messages in a pre-defined order and update each node immediately. Assume we have a two-dimensional grid with standard four neighborhood. Then we can for instance process the nodes along each direction separately while always using the newest message values during the computation.

   Another possibility is to interleave the update directions. Starting at the first node of a row, we compute the messages from left to right until we reach the last node of the row. Then instead of repeating the same pattern for the next row, we traverse the same row in opposite direction. Once we return to the first node, we have completed the row and move on to the next one. After all rows have been processed, we use the analogous scheme for the columns [28].

According to [29] the asynchronous update scheme is significantly faster than the synchronous version. We contribute this to the number of steps that we need to pass a message within the image. In the synchronous version, a message update can only cover one node per iteration, whereas in the asynchronous version updates can move across the whole image in a single iteration [29].

In practice, we can also improve stability and convergence speed by interpolating a message value between the old and the new message value. We model this compromise through an *inertia factor* α that computes the new message value as

$$m_{\text{new}} = \alpha\, m_{\text{update}} + (1 - \alpha) m_{\text{old}}$$

where m_{update} denotes the newly computed message value, m_{old} the old message value and m_{update} the value that we finally assign as new message values [38].

The properties of the belief propagation algorithm are twofold. On the one hand it is proven to efficiently compute the exact MAP and MMSE estimates in tree-like graphical models. It can also be shown that the fixed points of the belief propagation algorithm are equivalent to an approximate free energy, known as the Bethe free energy in statistical physics. In numerous applications belief propagation delivers satisfactory approximations in acceptable execution time. Examples are turbo codes, free energy approximations, low density parity check codes, computer vision problems or medical diagnosis [37].

On the other hand belief propagation demonstrates poor convergence for graphical models with many tight loops and rather uniform potential functions. Even if the algorithm tends to the stationary points of the Bethe free energy, we are not guaranteed to obtain satisfactory results. For many images this approximation is rather imprecise and therefore unsuited as a convergence goal. We generally assume that the belief propagation algorithm delivers better results if it converges in a small number of steps [24]. Some empirical results for the convergence behavior of belief propagation are given in [24]. They report that the algorithm converged on two synthetic graphical models named PYRAMID and toyQMR as well as on one real-world example called ALARM, but failed to converge on the QMR-DT example, which is a large bipartite network with noisy-ORs as partition functions. In [39] the authors experimented with a square lattice Ising spin glass model of different lengths. They instantiated all data and smoothness partition functions with random and independent values and experienced that standard belief propagation already fails to converge for grids of length 20. Considering the small size of this network, this result gives cause for concern. We may conclude that belief propagation suffers from serious stability problems in many real-world networks of large complexity, especially in the three dimensional case.

However, in the same paper the authors analyzed the performance of a modified belief propagation algorithm which they baptized *generalized belief propagation*. In comparison to belief propagation its results are very promising. On the spin glass model of size 10 for instance this algorithm finds nearly the exact estimates, whereas the belief propagation algorithm leads to completely exaggerated values. What is the idea of the GBP algorithm?

4.4 Generalized belief propagation

The standard belief propagation algorithm features many attractive properties, such as exact estimates in trees or fast execution time if it converges. But it also shows poor convergence for many looopy graphs that manifest themselves in inaccurate estimates. The source of error is the circular message flow that distorts good message approximations by self-dependent probability values. However, [39] proposes a way to alleviate this undesired behavior. The basic idea is to compute more informative messages between node regions in addition to messages between single nodes. What we obtain, is an algorithm that demonstrates improved convergence behavior and delivers very accurate approximations to the exact estimates. It no longer tends to the stationary points of the Bethe energy but is proven to approximate the fixed

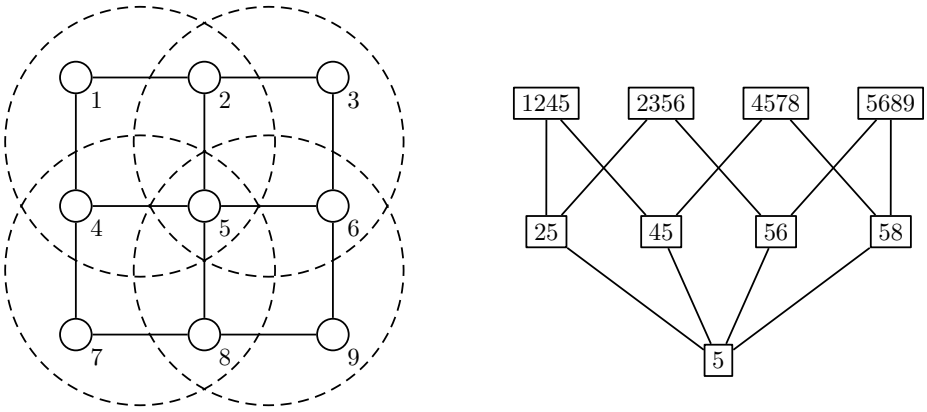


Figure 4.5: On the left we see a diagrammatic representation of the basic clusters; on the right we see a possible region graph for the corresponding Markov random field.

points of the more precise Kikuchi energy [37]. The consistent name of the algorithm is *generalized belief propagation (GBP)* [39]. Let us explore the modifications in detail.

The first variation emerges during the initialization. In contrast to the standard BP algorithm, the GBP algorithm expects us to mark *basic clusters* that are necessary for defining message update rules between group of nodes. Unfortunately, it is an art to choose the optimal cluster size [33]. The problem is that the basic clusters should encompass as many cycles as possible, but must not be too large since the GBP algorithm grows exponentially with the second largest cluster size. In practice, cluster sizes larger than four may already be infeasible. An example for basic clusters is depicted in figure 4.5. The clusters tightly capture the smallest loops of the graph and are still of reasonable size. Having defined a set of basic clusters, we continue the construction process by determining sub-regions. For any two basic clusters we compute the intersection and define non-empty sets as sub-regions. In 4.5 for example 45 is a subregion of the basic clusters 1245 and 4578. Then we intersect sub-regions to gain the next generation of clusters. We repeat this pattern recursively until the intersections equal single nodes. In example 4.5, we obtain for instance the final cluster 5 out of 45 and 56.

For visualizing the intersection dependencies among all these regions, we use a succinct representation called *region graph*. A region graph is a directed acyclic graph, where vertices correspond to node regions and edges are only permitted between a region P and a direct sub-region C if C contains all or a subset of the nodes in P. We say that P is a *parent region* of C because a directed edge leads from P to C. Conversely, we denote C as a *child region* of P. If there is a directed path from a region A to C, A is called an *ancestor region*, whereas a region D that is reached from C through a directed path is named *descendant region*.

Finally, a region graph has to satisfy a technical requirement called *region graph condition*. Suppose we like to estimate the marginal probability of a large region, e.g. the entire graphical model, which is as we know intractable for complex networks. Then the idea of the GBP algorithm is to define feasible sub-regions that altogether cover the large region. But as the sub-regions probably overlap, we cannot simply calculate the sum. We have to ensure that each variable of the large region is only

considered once. For this reason we associate a weight or *counting number* with each region R such that we fulfill the *region graph condition*

$$\sum_{R \in RG(s)} c_R = 1 \quad \text{for all } s \in S$$

where $RG(s)$ denotes the set of regions containing variable s [33].

In figure 4.5 we see a Markov random field and one suitable region graph. We observe that it represents each cluster generation in a distinct level and connects a region R to a sub-region S if S arose from an intersection step between R and another node [39].

We can distinguish three ways of representing messages and beliefs, each showing different strengths:

1) *The region graph*: The region graph is a good choice for understanding which messages enter belief regions and have influence on certain message computations.

2) *Diagram of the graphical model*: A diagrammatic representation of a graphical model is suited to visualize message and belief dependencies for grid graphs with consistent neighborhood systems.

3) *Formulas*: The belief formula and the message update rule are necessary for a mathematical sound and compact description of the GBP algorithm. However, formulas may be inappropriate for illustrating the idea behind the algorithm.

With these three representation types at our disposal we like to comprehend how the GBP algorithm composes the message update rule. As in the BP algorithm we derive the general formula by combining the marginalized condition

$$b_R(x_R) = \sum_{x_A \setminus x_R} b_A(x_A)$$

with the belief expressions $b_R(x_R)$ and $b_A(x_A)$, where region A has to be the parent of region R in the region graph.

We consider *belief* at a region R to be proportional to the product of local data and smoothness potential functions, multiplied by messages entering region R from outside and messages that lead from not-descendant regions of R into descendant regions of R. Only this composition of messages is proven to minimize the free energy of the region graph. The formal definition of *beliefs* is written as

$$b_R(x_R) = k\psi_R(x_R) \left(\prod_{P \in \mathcal{P}(R)} m_{P \to R}(x_R) \right) \left(\prod_{D \in \mathcal{D}(R)} \prod_{P' \in \mathcal{P}(D) \setminus \mathcal{E}(R)} m_{P' \to D}(x_D) \right)$$

where k normalizes the sum of beliefs to 1 and $\psi_R(x_R)$ denotes all potential functions included in region R. The variable $\mathcal{P}(R)$ describes the set of regions that are parent of region R (according to the region graph representation), $\mathcal{D}(R)$ is the set of descendant regions, and $\mathcal{E}(R) \equiv R \cup \mathcal{D}(R)$ comprises region R and all its descendant regions.

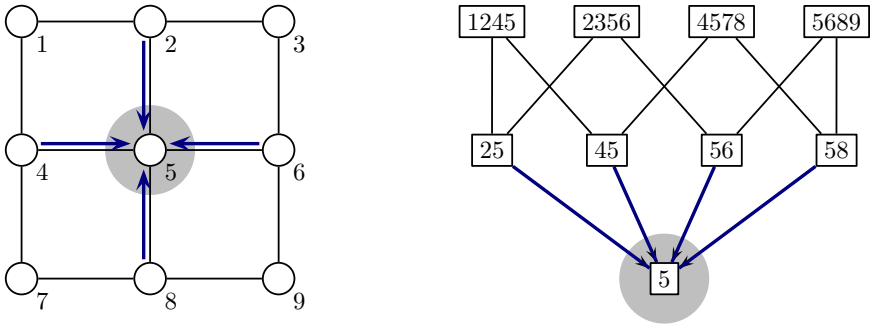


Figure 4.6: On the left we see a diagrammatic representation for a single-node belief region, highlighted with a gray background. Its formula is $b_5 = k\,\phi_5\, m_{25} m_{45} m_{65} m_{85}$. The right figure depicts the corresponding region graph. Blue directed edges denote messages between single nodes.

Consequently, $\mathcal{P}(D)\backslash\mathcal{E}(R)$ denotes the set of all parent regions of D except R or any of R's descendant regions [38].

After substituting the belief expressions into the marginalized condition, we retrieve the *message update rule*:

$$m_{P\to R}(x_R) = \frac{\sum_{x_{P\backslash R}} \left(\psi_{P\backslash R}(x_{P\backslash R}) \prod_{(I,J)\in N(P,R)} m_{I\to J}(x_J)\right)}{\prod_{(I,J)\in D(P,R)} m_{I\to J}(x_J)}$$

where $N(P,R)$ denotes the set of all connected pairs of region (I,J) such that I is not in $\mathcal{E}(P)$ while J is in $\mathcal{E}(\mathcal{P})$ but not in $\mathcal{E}(R)$. The set $D(P,R)$ stands for the set of all connected pairs of regions (I,J) such that $I \in \mathcal{D}(P)$ but not in $\mathcal{E}(R)$ whlie J is in $\mathcal{E}(R)$. Note that we can precompute $N(P,R)$ and $D(P,R)$ for improving the execution time of the GBP algorithm. Figure 4.6 to 4.9 may clarify the ideas of both formulas.

We should remark that it is possible to construct GBP algorithms that minimize the same free energy by relying on alternated region graphs. For example, we could introduce arcs in the region graph that lead from grandparent regions to its grandchildren. Even for the same region graph we can retrieve different versions of the GBP algorithm. For our definition of the region graph, we could choose among at least three variants that all propose slightly different formulations for the belief equation and the message update rule: the *parent-to-child algorithm*, the *child-to-parent algorithm*, and the *two-way algorithm*. We decided to present the *parent-to-child algorithm* because its definition does not contain any region counting numbers, just as our formulation of the BP algorithm [38].

We may ask ourselves how much we gain by evaluating these more intricate definitions of the belief equation and the message update rule. Concerning the accuracy and stability, we can claim that the GBP algorithm clearly outperforms the BP algorithm. On tree-like graphs, the algorithm finds the exact estimate, just as the BP algorithm; on cyclic network the GBP algorithm even converges in less iterations to

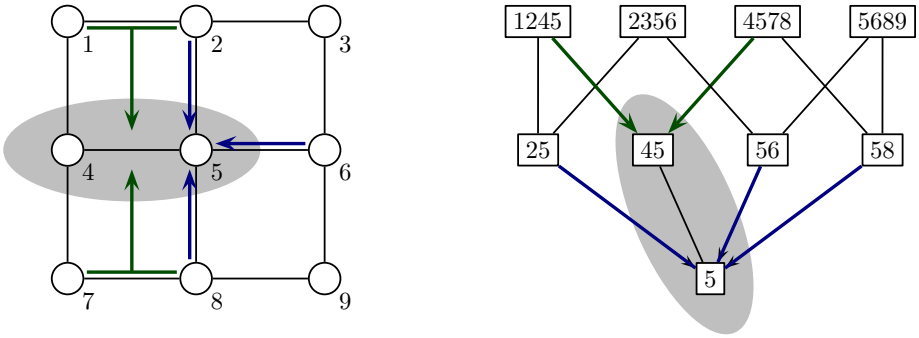


Figure 4.7: On the left we see a diagrammatic representation for a two-node belief region. Its formula is $b_{45} = k\,\phi_4\phi_5\psi_{45}\,m_{25}m_{65}m_{85}m_{1245}m_{7845}$. The right figure depicts the corresponding region graph. Green directed edges denote messages between edges, i.e. two-node beliefs.

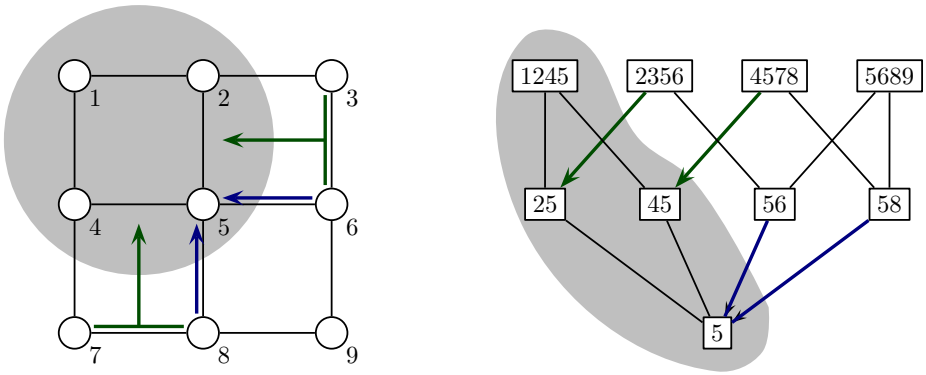


Figure 4.8: On the left we see a diagrammatic representation for a four-node belief region. Its formula is $b_{1245} = k\,\phi_1\phi_2\phi_4\phi_5\psi_{14}\psi_{25}\psi_{45}\,m_{65}m_{85}m_{3625}m_{7845}$. The right figure depicts the corresponding region graph.

approximative estimates and generally with much higher probability. The precision of the estimates is no coincidence. As [37] proves, the fixed points of the GBP algorithm correspond to the minima of the Kikuchi energy, which is a better approximation to the free energy than Bethe. Again, we observe that a more accurate approximation of the underlying free energy coincides with an improved stability of the algorithm.

Unfortunately, we cannot guarantee that the algorithm converges. Although the literature [24] [41] suggests that it should be reliable enough for most vision problems, it may fail to converge. In these rare cases, we can exploit that the GBP algorithm may iterate forever while the minima of the Kikuchi/Bethe free energy deliver a fairly accurate estimate for the exact free energy. Prominent examples for algorithms that directly minimize the free energy are: *CCCP* [41], *UPS* [31] or the *double-loop algorithm* [15]. What are their major ideas?

The *concave convex procedure (CCCP)* decomposes the free energy into concave and convex parts to formulate discrete update rules that iteratively decrease the free

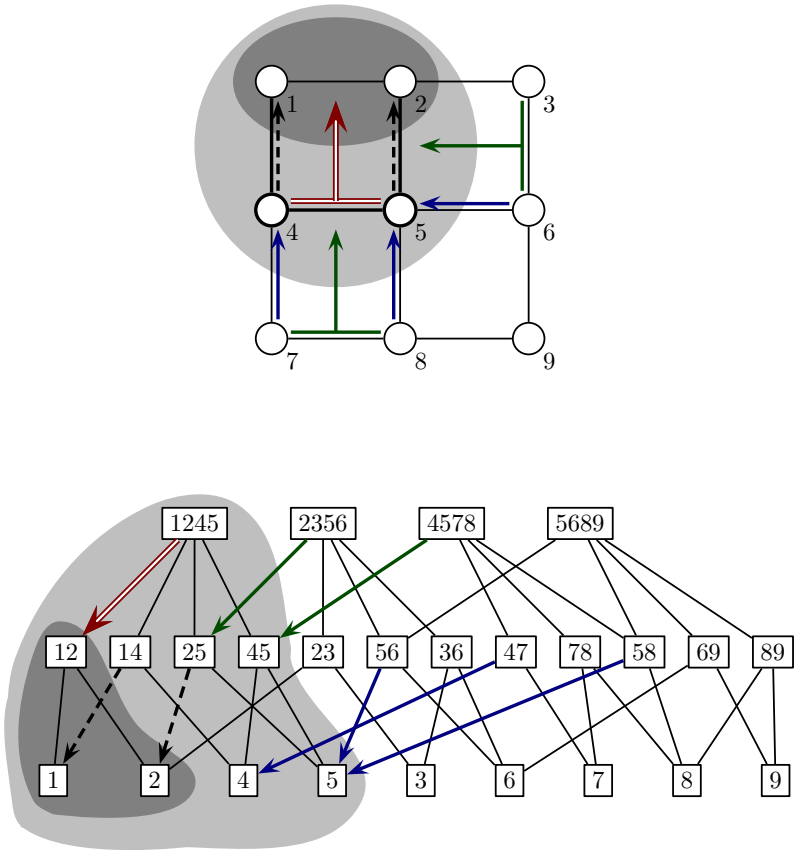


Figure 4.9: In the upper figure we see a diagrammatic representation of the GBP message update rule for a cluster message. The source belief region of the message is depicted in light gray, whereas dark gray indicates the target belief region. The red directed edge denotes the cluster message m_{4512} which leads from edge 45 to edge 12. Its formula is $m_{4512} = \phi_1\phi_2\phi_4\phi_5\psi_{12}\psi_{14}\psi_{25}\psi_{45}\, m_{74}m_{65}m_{85}m_{3645}m_{7845}$. Below it we have depicted the complete region graph as it is commonly constructed in the GBP algorithm.

energy. Very similar is the proceeding of the *unified propagation and scaling algorithm (UPS)*. Both algorithms are guaranteed to converge against the extremum of the Bethe free energy and are more likely to converge against the Kikuchi energy than the standard GBP algorithm. The main difference between both approaches lies in the convergence rates [30]. A third alternative is the *double-loop algorithm*. It tries to solve our non-convex constrained optimization problem by minimizing convex bounds on the Kikuchi free energy. According to the authors this method yields dramatic speed-ups compared to the very slow CCCP and UPS algorithm. Nevertheless, we will follow the standard GBP algorithm, as it converges for most the problems and more importantly its runs faster than the very slow *double-loop algorithm*.

This argument carries even more weight when stating that speed is the major drawback of the GBP algorithm. In contrast to the BP algorithm, we have to perform a vast number of additional computations per iteration. Consider the complexity of two- and three-dimensional images represented as Markov random field in standard form.

1) *2D image*: If we represent an image of size $w \times h$ by a two-dimensional grid graph with standard four neighborhood and assume basic clusters of size four, we have to evaluate

$$4\lfloor w-1 \rfloor \cdot \lfloor h-1 \rfloor$$

messages between edge-like node regions in addition to $4wh-2w-2h$ messages between single nodes. This means roughly twice as many messages as in the BP algorithm.

2) *3D image*: In a three-dimensional image $w \times h \times d$, we even have to calculate the values of

$$4\left(w\lfloor h-1 \rfloor \lfloor d-1 \rfloor + \lfloor w-1 \rfloor h \lfloor d-1 \rfloor + \lfloor w-1 \rfloor \lfloor h-1 \rfloor d\right)$$

edge messages apart from the $6whd-2wh-2wd-2hd$ message values between single nodes. Hence, nearly three times as many messages as in the standard BP algorithm.

Even worse is that each evaluation of the belief expression and the message update rule makes reference to messages between groups of nodes. Unlike messages between single nodes, we cannot represent them through vectors. We have to use matrices whose number of dimensions equals the size of the largest intersection region in the region graph. The matrices that are related to edge messages are of size k^2 where k denotes the number of possible labels. Of course, it is possible to reduce the computational costs if the message values correspond to special matrices, e.g. symmetric or uniform matrices, but for non-trivial networks we generally struggle with unbearable execution times.

We can summarize that the standard form of the GBP algorithm converges fairly reliably to accurate approximations of the estimates and can be flexibly adjusted to our vision problem. However, it is computationally too expensive for many applications. The decisive question is if we can *accelerate* the GBP algorithm.

In the literature, we have spotted only two papers, [27] and [19], that focus on speedup techniques for the GBP algorithm, and both are guided by the same idea: The most common pairwise potentials can be divided into *compatible* pairs of labels whose values are label-dependent and *incompatible* pairs of labels that all take the same value. As the number of compatible pairs of labels n_c is usually much smaller than the number of incompatible labels n_i, we gain a speed-up of n_c/n_i by not computing redundant incompatible labels. Shental et al. [27] suggests this approach for the Ising model, and Kumar et al. [19] for the more general *robust truncated model*, comprising the piecewise constant prior and the piecewise smooth prior.

To motivate further acceleration techniques, we may briefly review recent work for the similar BP algorithm. Felzenswalb et al. [8] present three acceleration techniques for grid graphs and require the pairwise potential to be the Potts model, the linear, or the quadratic truncated model. They first observe that each message update rule can be expressed as a *min convolution* [7] which allows them to reduce the complexity from quadratic to linear in the number of possible labels. Thus, the concept resembles the approaches of [27] and [19]. Second, they manage to halve the number of message updates. Similar to a checkerboard pattern, they divide the grid graph into two sets of nodes and alternate the message updates for each set. And third, they attempt to

initialize the message close to a fixed point for reducing the number of iterations until convergence. Their idea it to apply the BP algorithm in a coarse-to-fine multi-grid and initialize the messages of finer levels with the approximations of the next coarser level. A nice property of all three techniques is that they deliver the same approximations as the standard form of the BP algorithm. In [20] the authors generalize the first technique to belief propagation on arbitrary Markov random fields. [34] proposes a coarse-to-fine multi-grid in which edges within a level are partly replaced by edges between levels. The disadvantage of this approach, however, is that it changes the minimization problem and does not necessarily lead to the same solution.

In this thesis, we attempt to apply four different acceleration techniques, from which three are novel and one is an adaptation of an available methods.

Chapter 5

Speedup techniques

5.1 Overview

In this chapter, we discuss four algorithmic ideas for accelerating the standard GBP algorithm: (1) *hierarchical initialization*, (2) *following active messages*, (3) *caching and multiplication* and (4) *acceleration for MAP estimate*. The first technique is adapted from a paper of Felzenswalb et al. [8] which covers speedup techniques for the standard BP algorithm; the remaining three techniques are novel to our knowledge.

With the exception of the second technique all approaches assume that the underlying graphical model is a two-dimensional grid with a four-connected neighborhood system, respectively a three-dimensional grid with six neighbors. This may sound like a severe restriction, but for many computer vision problems it is not. Grid graphs are the common representation for images and four- respectively six-connected neighborhood systems which we shall use throughout this chapter are widespread as well. Likewise, we do not care that the forth technique is solely effective on labeling problems for which we search the MAP estimate, since it is our preferred choice (cf. section 4.1.4).

Apart from these limitations the techniques incorporate some very pleasant properties. For instance, they all work on arbitrary energy functions. Unlike [19] and [27] we do not require that the energy function exhibits a beneficial structure. Thus, we may flexibly design the energy function that best suits our labeling problem and can still expect to benefit from significant speedups. Another advantage is that all techniques are fully compatible to each other, i.e. we can exploit them simultaneously. The reason is that they either leverage distinct parts of the GBP algorithm, or they neatly complement each other. To make this more explicit, we may briefly describe each technique with the help of the specialized GBP algorithm for grid graphs:

1) *Hierarchical initialization*: Instead of initializing messages with uniform values, we hope to gain significant speed-ups by adapting the multi-grid approach of [8] for two-and three dimensional grid graphs with standard four-connected, respectively six-connected neighborhood system. The technique exclusively affects line 1-3 of the algorithm.

2) *Following active messages*: Our first novel approach is directed towards reducing the number of visited messages in each iteration. We attempt to compute

Algorithm 2 GBP algorithm for grid graphs

Require: Positive thresholds ϵ and $iter_{\max}$.

for all messages m_i **do**
  $m_i^{\text{old}} \Leftarrow 1$ ▷ (1) Hierarchical initialization
end for
for all single-node beliefs b_i **do**
  $b_i^{\text{old}} = 1$
end for
$d \Leftarrow 1000$
$iter \Leftarrow 0$
while $(d \geq \epsilon)$ and $(iter < iter_{\max})$ **do**
  for all edge messages m_i **do** ▷ (2) Active message technique
    Compute new message value m_i^{new} ▷ (3) Caching and Multiplication, ▷ (4) Accelerating MAP estimate
  end for
  for all cluster messages m_i **do** ▷ (2) Active message technique
    Compute new message value m_i^{new}. ▷ (3) Caching and Multiplication, ▷ (4) Accelerating MAP estimate
  end for
  $d \Leftarrow 0$
  for all single-node beliefs b_i **do**
    Compute new belief b_i^{new} with the help of messages m^{new}.
    $d \Leftarrow d + |b_i^{\text{new}} - b_i^{\text{old}}|$
    $b_i^{\text{old}} \Leftarrow b_i^{\text{new}}$
  end for
  for all messages m_i **do**
    $m_i^{\text{old}} \Leftarrow m_i^{\text{new}}$
  end for
  $iter \Leftarrow iter + 1$
end while
for all nodes n_i **do**
  $\text{label}(n_i) \Leftarrow \arg\max_k b_i(k)$
end for

only "active" messages, i.e. messages that have not converged yet. The algorithm comes in two different flavors: First, we present a conservative heuristic that guarantees the same solution as the standard form of the GBP algorithm. Second, we design a heuristic that trades accuracy for speed and can be advantageous if we are not interested in the exact probabilities for each label of a node. The technique aims at line 10 and 13 of the algorithm.

3) *Caching and multiplication*: We introduce a novel technique that pre-calculates interim message products used by multiple messages and optimizes the product order of message update rules. Consequently, the technique pertains to line 11 and 14.

4) *Acceleration for MAP estimate*: The last novel technique also affects the message update rules in line 11 and 14. In contrast to the preceding technique, though, it concentrates on the inner workings of the multiplication operation.

In the rest of this chapter, we describe the idea of each technique in more detail and provide some theoretical estimates about the expected speedup. Admittedly, this is not always possible but our complexity considerations may contribute to understand and interpret the experimental results at the end of this thesis.

5.2 Hierarchical initialization

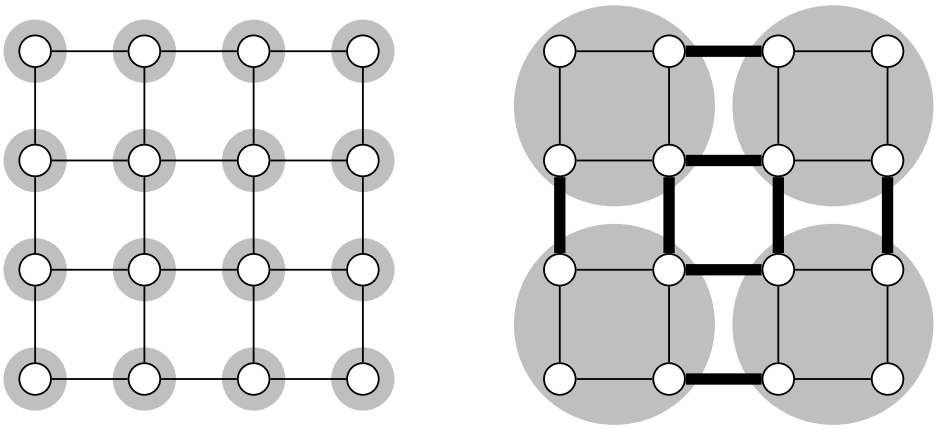
The first step of the GBP algorithm is the initialization of messages to random or uniform values. We observe that both choices are suboptimal because they are far apart from the fixed points, which reduces the convergence speed of the GBP algorithm. Thus, we could accelerate the GBP algorithm, if we initialized the message values to better approximations of the stationary points.

One approach for obtaining a good approximation stems from Felzenswalb et al. [8]. They describe a multi-grid approach for the standard BP algorithm that we like to adapt for the GBP algorithm. The basic idea works in two principle steps: (1) We construct a hierarchy of increasingly coarsened versions of an image. (2) Starting at the coarsest level l, we run the GBP algorithm until it converges and use its estimates to initialize the messages at the next finer level $l-1$. This strategy is repeated until we reach level 0 where we can hopefully initialize the messages close to the fixed point.

The GBP algorithm benefits from a multi-grid Markov random field by allowing messages to travel much faster over long distances. Depending on the hierarchy level, messages can proceed in various step sizes, which is not possible for mono-grids. [17]. Willsky [34] for instance extends the Markov random field by potential functions to coarser hierarchy levels, thereby a hierarchical Markov random fields to

5.2.1 Hierarchical initialization in 2D

Let us define the algorithm more rigorously for a two-dimensional grid graph. We denote the sites of level i by S^i where $S^0 = S$ stands for the original grid. Each node k on level i refers to a block b_k^i on level 0 with size $2^i \times 2^i$. The intuition is that each labeling of a node on level i is tantamount to associating all nodes of the



(a) level 0 (b) level 1

Figure 5.1: Diagrammatic representation of the first two hierarchy levels. The highlighted regions indicate the nodes on the respective level and simultaneously blocks b_k that comprise the contained nodes of level 0. The bold edges are updated before the normal edges during the propagation update scheme.

corresponding block with the same label. Let us define the set of blocks on level i by

$$\mathcal{B}^i = \{b_1^i, \ldots, b_{n_i}^i\}, \qquad n_i = \frac{n}{4^i}$$

where n denotes the number of labels on the original grid. In figure 5.1, we can see an example for the two finest hierarchy levels of a multi-grid. The gray circles in the background indicate the blocks.

The energy function on hierarchy level i that shall be minimized by the GBP algorithm is given by

$$\begin{aligned} E(x^i) &= E_{\text{data}}(x^i) + E_{\text{smooth}}(x^i) \\ &= -\sum_{b_k^i} w_k^{\mathcal{B}^i} V_k^{\mathcal{B}^i}(x_k^{\mathcal{B}^i}) - \sum_{b_k^i \sim b_l^i} w_{kl}^{\mathcal{B}^i} V_{kl}^{\mathcal{B}^i}(x_k^i, x_l^i) \end{aligned}$$

where x^i denotes the image at resolution level i. The pivotal question is how we define the *data potential* $V_k^{\mathcal{B}^i}$ and the *smoothness potential* $V_{kl}^{\mathcal{B}^i}$ at each hierarchy level.

In accordance to the proposition in [8] we define the *data potential* of block b_k^i as the sum of the data potentials for all nodes contained in the block:

$$V_k^{\mathcal{B}^i}(x_k^{\mathcal{B}^i}) = \sum_{s \in b_k^i} V_s(x_s)$$

This is equivalent to computing the product of the corresponding *data partition functions*

$$\phi_k^{\mathcal{B}^i}(x_k^{\mathcal{B}^i}) = \exp\left(w_k^{\mathcal{B}^i} V_k^{\mathcal{B}^i}(x_k^{\mathcal{B}^i})\right) = \exp\left(w_k^{\mathcal{B}^i} \sum_{s \in b_k^i} V_s(x_s)\right) = \prod_{s \in b_k^i} \phi_s(x_s)$$

This means that the data potential of the block is set to the probabilities of observing the same label for all its contained nodes. We remark that the summation of the data potentials can be recursively done from bottom to top, i.e. we solely have to sum over the corresponding four data potentials of the next finer level [8].

The *smoothness potential* on level i typically depends on the block size ϵ and the potential candidate that we choose for the labeling problem. For example, we can model the smoothness potential that varies with the difference of the labels with a linear truncated model depending on ϵ:

$$V_{kl}^{\mathcal{B}^i}(x_k^{\mathcal{B}^i} - x_l^{\mathcal{B}^i}) = \min\left(\epsilon V_{kl}\left(\frac{x_k^{\mathcal{B}^i} - x_l^{\mathcal{B}^i}}{\epsilon}\right), d\right)$$

where we divide by ϵ to reflect the smaller influence between distanced blocks and multiply with ϵ to model the number of neighboring sites along the boundary. The variable d represents a pre-defined truncation factor [8].

Note that it is generally task of the computer vision specialist to model the smoothness potential for each hierarchy level.

So far, we discussed how we model potential functions of the message update rule in the GBP algorithm. The next step is to propose an advantegeous *message propagation scheme* from coarse to fine levels. In the case of two-dimensional grid graphs we proceed in five steps:.

Algorithm 3 Message propagation scheme on 2D grid graphs

1: Copy the edge message estimates between two blocks to all messages that connect these blocks on the next finer level.
2: Propagate the cluster message value between four blocks at level l to level $l-1$.
3: Compute the residual edge messages where we ignore message terms that have not been initialized yet.
4: Compute cluster messages at the border of blocks where we ignore message terms that have not been initialized yet.
5: Compute the residual cluster messages.

The advantage of the update scheme is that we can incorporate more and more initialized message terms in the message update rules. We consecutively guide the initialization from the known message values of the upper level to unconfigured messages of the current level. Figure 5.1 may help to examine the proceeding. We begin with edge messages in bold and cluster messages that are situated in regions with a complete bold border. Next we compute the residual edge messages. And finally, we devote ourselves to cluster messages which are surrounded by exactly two bold edges.

Let us analyze the complexity of the multi-grid approach. Except for the finest hierarchy level we abort the GBP algorithm after a couple of iterations. As an aid to orientation, we expect that the GBP algorithm needs less iterations than the standard BP algorithm, which is reported to deliver a sufficient approximation within five to ten iterations [8]. The memory cost for the additional hierarchy levels amounts to less than $\frac{1}{3}$ of the number of nodes in the finest level, since we can estimate

$$\sum_{i=0}^{l} n_i < \sum_{i=0}^{\infty} n_i = n\sum_{i=0}^{\infty}\frac{1}{4^i} = n\frac{1}{1-\frac{1}{4}} = \frac{4}{3}n$$

where l denotes the number of hierarchy levels.

5.2.2 Hierarchical initialization in 3D

Felzenwalb's multi-grid technique also scales to three-dimensional images if we slightly adjust the size of the blocks and the message propagation scheme. As before, each node k on level i is associated with a *block* b_k^i on level 0, but with alternated size $2^i \times 2^i \times 2^i$. We can express the set of blocks on by

$$\mathcal{B}^i = \{b_1^i, \ldots, b_{n_i}^i\}, \qquad n_i = \frac{n}{8^i}$$

The energy function and the considerations on the partition functions can be reapplied without change from the two-dimensional case. However, we have to modify the second step of the *message propagation scheme*:

Algorithm 4 Message propagation scheme on 3D grid graphs

1: Copy the edge message estimates between two blocks to all messages that connect these blocks on the next finer level.
2: Propagate the cluster message value between four blocks at level l to level $l-1$.
3: Compute the residual edge messages where we ignore message terms that have not been initialized yet.
4: Compute cluster messages at the border of blocks where we ignore message terms that have not been initialized yet.
5: Compute cluster messages within blocks.

We attempt to initialize the messages as close as possible to the stationary points. As depicted in figure 5.2, we proceed by using as many initialized incoming messages as possible in the message update rules. As for the two-dimensional case bold edges are initialized before normal edges. With regards to cluster messages we first initialize cluster messages that are enclosed by bold lines, for instance the green region. Then, we pass on to the adjacent cluster regions, colored in yellow and finally compute cluster messages within blocks, such as the red region.

5.3 Active-message technique

5.3.1 Exact variant

The *active-message technique* is a strategy to steadily reduce the number of computed messages per iteration. The fundamental observation is that the convergence test of the standard BP and the GBP algorithm is determined by a decreasing number of belief differences. Messages differ in the number of iterations that they need to stabilize at their equilibrium. Thus, the idea of the algorithm is to save message computations that are considered to be stable and to proceed solely with changing, i.e. *active* messages. The criterion for distinguishing active from inactive messages is a simple threshold τ. Empirical results show that it is wise to choose a value close to zero since even small message differences may add up significantly over several iterations, enough to corrupt the estimates.

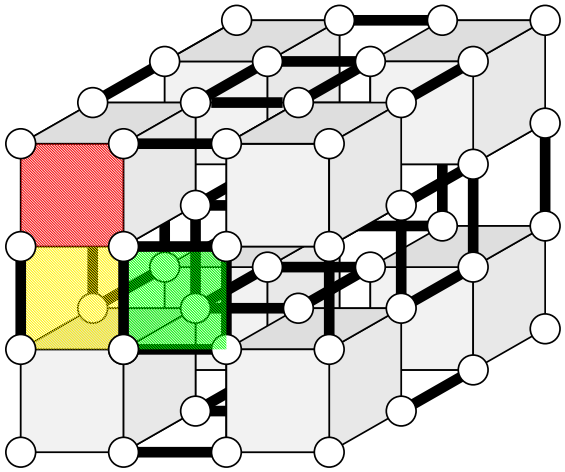


Figure 5.2: A diagrammatic representation of the hierarchical initialization in a three-dimensional grid graph. The gray regions correspond to blocks in the observation field, respectively nodes on the next coarser level. Bold edges indicate messages that are evaluated before normal edge messages. The green cluster message is an example for the type of cluster message that is preferably evaluated. Second are messages that are bordered by two bold lines, depicted in yellow. Finally, we calculate cluster messages within blocks, such as the red region.

Can we be certain that this strategy tends to the same solution as the standard GBP algorithm? The answer is no. As we consider all active messages to be a subset of the active messages in the preceding iteration, we do not capture messages that change outside the candidate set after one or multiple iterations. Sometimes, the number of missed active messages is rather small and does not significantly influence the estimates from our technique but until now we have not discovered the exact circumstances under which this is the case. One step, however, is certainly necessary if we guarantee the same result as the standard GBP algorithm: We have to run the standard GBP algorithm after our technique converged.

Let us express the *active-message technique* in pseudo-code notation (cf. algorithm 5).

5.3.2 Approximate variant

In labeling problems we are generally not interested in the exact probabilities of each possible label. It suffices to search for the label with the highest probability. This means for the active-message technique that we can relax the selection criterion for active messages, provided that the algorithm ultimately votes for the same label configuration. Opposed to the exact variant we can consider messages to be inactive that change more than τ but do not affect the label of the involved nodes. Conversely, messages between stable labels are neglected in future iterations. But how do we formulate a heuristic that reliably and efficiently predicts as many inactive labels as possible?

We have the label history and the probabilities for each node at our disposal. As

Algorithm 5 Active-message technique (exact variant)

1: Run the standard GBP algorithm for two iterations to gain a candidate set $\mathcal{A}$ of *active messages*.
2: **while** $(d \geq \epsilon)$ and $(iter < iter_{\max})$ **do**
3: **for all** *active* messages $m_i^{\mathcal{A}}$ **do**
4: Compute new message value $(m_i^{\mathcal{A}})^{\text{new}}$
5: $d_i^{\mathcal{A}} \Leftarrow |(m_i^{\mathcal{A}})^{\text{new}} - (m_i^{\mathcal{A}})^{\text{old}}|$
6: **if** $d_i^{\mathcal{A}} < \tau$ **then**
7: Remove $m_i^{\mathcal{A}}$ from $\mathcal{A}$.
8: **end if**
9: Compute beliefs that depend on active messages and update the belief difference d analogous to the standard GBP algorithm.
10: **end for**
11: **end while**
12: Run the standard GBP algorithm until it converges.

we strive for maximal speedup, it is important that the heuristic is based on label information that is as recent and local as possible. It would be too expensive if we first have to construct a long label history for each node to decide if the label has converged to its final value. Also, we attempt to perform the selection as fast as possible, constricting our label and probability evaluation to a small neighborhood.

Another open question is the selection of active messages flowing into or from an active belief region. This section can be considered as an outline of this acceleration technique. We have not yet found a reliable heuristic.

5.4 Caching and Multiplication

5.4.1 Caching and multiplication in 2D

If we assume that the graphical model equals a two-dimensional grid graph with a four-connected neighborhood system, we can explicitly compute the method for the grid graph. This spares us to laboriously traverse the region graph for computing the incoming messages in the belief and the message update formula.

Thus, for single-node beliefs we obtain the following (static) formula:

$$b_s(x_s) = k\phi_s m_{as} m_{bs} m_{cs} m_{ds}$$

where we use shorthand notation $m_{as} \equiv m_{as}(x_s)$ and $\phi_s = \phi_s(x_s)$. According to figure 5.3, the variables a, b, c and d denote the neighbors of s. To save time we do not compute beliefs for multi-node regions. In practice, we did not encounter any notable effects on the convergency of the algorithm.

The message update rule for *edge messages*, i.e. messages between two single nodes, is

$$m_{su}(x_u) = \max_{x_s} \left(\phi_s \psi_{su} m_{as} m_{bs} m_{cs} m_{bdsu} m_{cesu}\right)$$

where we abbreviate $\psi_{sx} = \psi_{sx}(x_s, x_u)$. Figure 5.4 shows a diagrammatic representation of the involved messages and potentials.

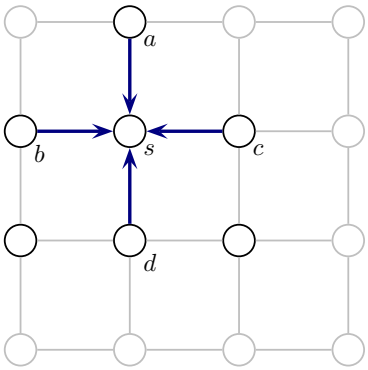


Figure 5.3: A diagrammatic representation of the messages that influence the single-node belief at site s in a two-dimensional grid.

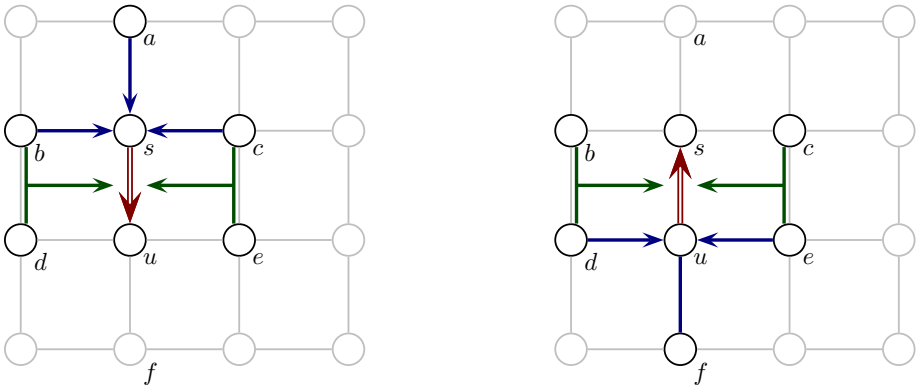


Figure 5.4: A diagrammatic representation of all edge messages (in red) that are contained in the same two-node belief region $R = \{s, u\}$. Note that the cluster messages from edges to edges are identical in both figures.

Cluster messages

The message update rule for *cluster messages*, i.e. messages between two pairs of nodes, unfolds as

$$m_{stuv}(x_u, x_v) = \frac{\max_{x_s x_t} \left(\phi_s \phi_t \psi_{st} \psi_{su} \psi_{tv} m_{as} m_{cs} m_{bt} m_{dt} m_{abst} m_{cesu} m_{dftv}\right)}{m_{su} m_{tv}}$$

Figure 5.5 depicts the potentials and messages that compose a cluster message.

If we apply these hard-coded formulas, we consider only values of incoming messages whose source and target nodes lie in the grid graph. Compared to the standard form, we avoid the initial recursive traversal of the region graph and need less memory, since we do not have to store references from a messages to its dependent messages.

If we scrutinize the messages that are computed within the same two- or four-node region, we notice that some messages appear repeatedly. This can for instance be observed in figure 5.5 which shows the four possible cluster messages within a basic cluster. Each cluster message computation involves four of the eight surrounding

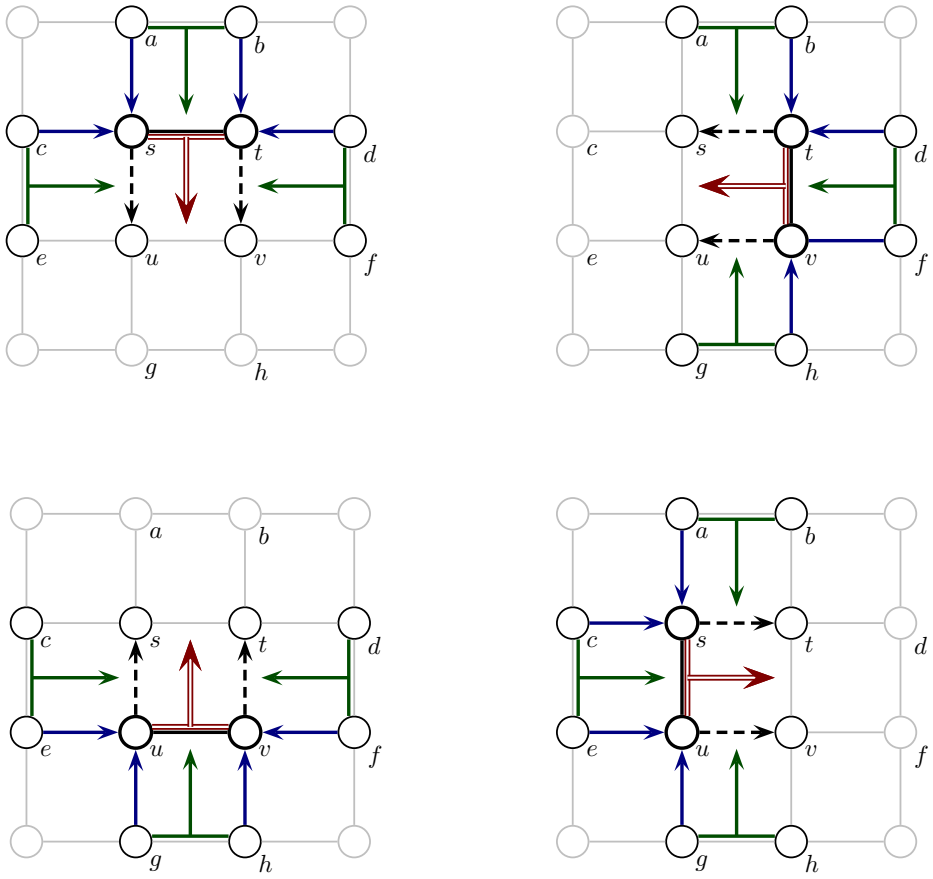


Figure 5.5: A diagrammatic representation of all cluster messages (in red) that are contained in the same four-node belief region $R = \{s, t, u, v\}$. Blue edges stand for edge messages in the nominator, whereas dashed edges are those in the denominator of the corresponding message update rule. Green messages denote as usual cluster messages that influence the value of the red cluster message. We can observe that messages appear within several figures.

edge messages and three of the four surrounding cluster messages. Remarkably, the selection of the messages is not arbitrary but follows a simple pattern. Let us analyze how the message update rule is composed for a cluster message m_{stuv}, where s and t are source nodes, and u and v are target nodes. Data potentials ϕ are only defined for the source nodes of m_{stuv}, while pairwise potentials ψ necessitate that at least one involved node is a source node. Similarly, incoming edge messages lead from outside the basic cluster to a source node of m_{stuv}, while incoming cluster messages demand that at least one of its source nodes has to be a source node of m_{stuv}.

Thus, the message update rule contains for each source node the product of all incoming edge messages and the data potential. We constitute the source node products as

$$P_s = \phi_s m_{as} m_{cs}$$

and

$$P_t = \phi_t m_{bt} m_{dt}.$$

Each product involves $2k$ multiplications, where k is the number of possible labels.

We continue to group factors in order to save operations. The product of the pairwise potential and the corresponding cluster message can be computed in k^2 operations, yielding

$$P_{st} = \psi_{st} m_{abst}$$
$$P_{su} = \psi_{su} m_{cesu}$$
$$P_{tv} = \psi_{tv} m_{dftv}$$

Substituting the expressions into the message update rule, we obtain

$$m_{stuv}(x_u, x_v) = \frac{\sum_{x_s, x_t} P_s P_t P_{st} P_{su} P_{tv}}{m_{su} m_{tv}}$$

The grouping of the nodes allows us to formulate an efficient algorithm for computing the four messages within a basic cluster.

Algorithm 6 Caching and Multiplication

1: We pre-compute the products at each node of the basic cluster using the same scheme as in P_s and at each edge of the basic cluster using the same scheme as in P_{st}.
2: **for all** cluster message within the basic cluster **do**
3: Select the two products for the source nodes and all products for the edges except the one that connects the two target nodes.
4: Calculate the message by first marginalizing each product and then multiplying in the same order as in the above formula.
5: **end for**

Note that the speedup not only consists of evaluating eight instead of twenty products. The major contribution stems from the order in which we marginalize and multiply. To clarify this point let us compare the number of multiplications in standard order to the number of multiplications in the optimized order. In each case we first marginalize each nominator term and then perform the necessary multiplications.

After the marginalization we obtain *scalars* for $\phi_s, \phi_t, \psi_{st}$ and m_{abst} and *vectors* for $\psi_{su}, \psi_{tv}, m_{cesu}$ and m_{dftv}. The number of multiplications for the standard order adds up to

$$k^2(8k^2 + k + 2) = 8k^4 + k^3 + 2k^2$$

We need two operations for multiplying the scalars ϕ_s, ϕ_t and ψ_{st}, k operations for multiplying the intermediate scalar with ψ_{su}, k^2 operations for computing the outer product between the intermediate vector with ψ_{tv}, and another k^2 operations for multiplying each subsequent term with the previous matrix. Altogether we have to compute k^2 different marginalizations.

In contrast, the optimized order requires

$$k^2(k^2 + k + 2) + k^2 + 2k = k^4 + k^3 + 3k^2 + 2k$$

operations for he product of the nominator terms and the pre-calculation of the interim products.

Thus, the caching techniques may result in a speedup of factor 8 for cluster messages.

Edge messages

For *edge messages* we can also use caching and optimize the order of multiplication. In analogy to the product scheme for nodes in basic clusters, we can calculate the product of a node in an edge region by

$$P_s = \phi_s m_{as} m_{bs} m_{cs}$$

where a, b and c denote the neighbors s, as figure 5.4 insinuates. The edge consists of the remaining three terms

$$P_{su} = \psi_{su} m_{bdsu} m_{cesu}$$

The message update rule consequently is written as

$$m_{su}(x_u) = \sum_{x_s} P_s P_{su}$$

For the standard order we need

$$6k^2$$

multiplications opposed to

$$k^2 + 2k + 3$$

operations for the optimized multiplication scheme and the pre-calculation of the interim products.

5.4.2 Caching and Multiplication in 3D

We can extend the caching and multiplication technique to three-dimensional grids with a six-connected neighborhood system. The only difference is that the products of nodes and edges involve more terms than in the two dimensional case. We start with the explicit formula for single-node beliefs on these grid graphs:

$$b_s(x_s) = k\phi_s m_{as} m_{bs} m_{cs} m_{ds} m_{es} m_{fs}$$

See figure 5.6 for checking the involved messages.

The message update rule for messages between single nodes is written out as

$$m_{su}(x_u) = \max_{x_s} \left(\phi_s \psi_{su} m_{as} m_{cs} m_{ds} m_{es} m_{is} m_{absu} m_{dfsu} m_{egsu} m_{ijsu}\right)$$

and is visualized in figure 5.7.

We calculate the message update rule between groups of nodes with

$$m_{stuv}(x_u, x_v) = \frac{\max_{x_s x_t} \left(\phi_s \phi_t \psi_{st} \psi_{su} \psi_{tv} M_1 M_2\right)}{m_{su} m_{tv}}$$

where

$$M_1 = m_{as} m_{es} m_{gs} m_{ms} m_{bt} m_{ft} m_{ht} m_{nt}$$

$$M_2 = m_{abst} m_{efst} m_{mnst} m_{acsu} m_{gisu} m_{mosu} m_{bdtv} m_{hjtv} m_{nptv}$$

The involved variables can be gleaned from figure 5.8.

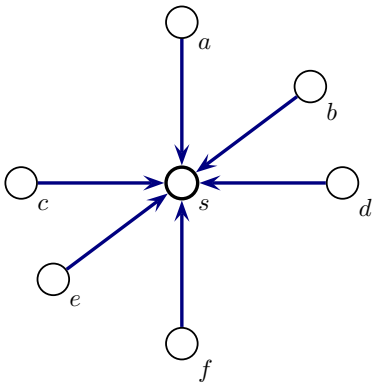


Figure 5.6: A diagrammatic representation of the messages that influence the single-node belief at site s in a three dimensional grid.

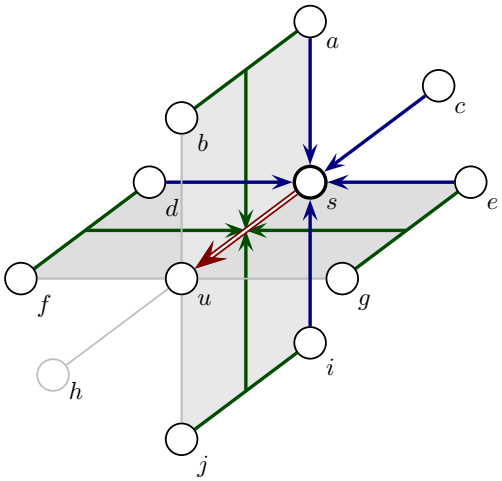


Figure 5.7: A diagrammatic representation of the messages that influence the edge message from site s to site u.

How does the caching technique work on cluster messages? Again, we define interim products on the source nodes of the cluster message:

$$P_s = \phi_s m_{as} m_{es} m_{gs} m_{ms}$$
$$P_t = \phi_t m_{bt} m_{ft} m_{ht} m_{nt}$$

Let us define the interim products on edges that contain at least one source node of the cluster message

$$P_{st} = \psi_{st} m_{abst} m_{efst} m_{mnst}$$
$$P_{su} = \psi_{su} m_{acsu} m_{gisu} m_{mosu}$$
$$P_{tv} = \psi_{tv} m_{bdtv} m_{hjtv} m_{nptv}$$

If we combine the products with the message update rule for cluster message, we

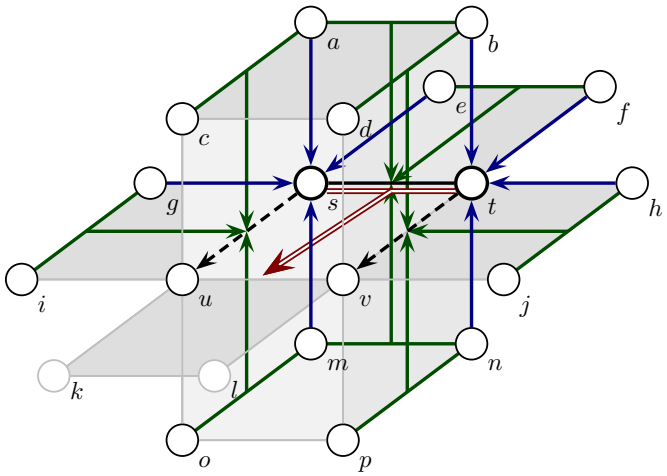


Figure 5.8: A diagrammatic representation of all messages that influence the red cluster message from edge st to edge uv. Edge messages in the nominator of the message update rule are depicted in blue. If they are in denominator they are dashed black. Green arrows refer to incoming cluster messages.

obtain the same formula as for the grid graph in two dimensions.

$$m_{stuv}(x_u, x_v) = \frac{\max_{x_s,x_t}(P_s P_t P_{st} P_{su} P_{tv})}{m_{su} m_{tv}}$$

In three dimensions, it turns out that the possible speed-up is even more drastic. The standard order of the cluster messages causes

$$18k^4 + 8k^3 + 2k^2$$

multiplications, whereas the optimized order only entails

$$k^4 + k^3 + 5k^2 + 4k$$

multiplications.

The interim product over the source variable of edge messages is computed by

$$P_s = \phi_s m_{as} m_{cs} m_{ds} m_{es} m_{is}$$

and the corresponding product over the edge follows from

$$P_{su} = \psi_{su} m_{absu} m_{dfsu} m_{egsu} m_{ijsu}$$

For the standard order we need

$$10k^2$$

multiplications opposed to

$$k^2 + 4k + 5$$

multiplications for the optimized order in

$$m_{su}(x_u) = \sum_{x_s} P_s P_{su}$$

5.5 Accelerating MAP estimation

Many computer vision problems can be stated as an energy minimization problem which we can solve by searching for the MAP estimate of the corresponding Markov random field. Thus, if we infer with generalized belief propagation, we have to use the *max-product* or equivalently the *min-sum* formulation.

Let us focus on the update rule for cluster messages in max-product form. It computes considerably slower than its edge message pendant, thereby offering a lot of potential to save time. The GBP algorithm grows with the power of four in the number of labels, whereas the computation of edge messages grows quadratic (cf. section 5.4).

Recall that we can write the update rule compactly (and efficiently) as

$$m_{stuv}(x_u, x_v) = (m_{su} m_{tv})^{-1} \max_{x_s, x_t} (P_s P_t P_{st} P_{su} P_{tv})$$

Until now we have analyzed in which order we should multiply the vectors and matrices within the message update rule. In this section, we pursue the question if it is necessary to exhaustively explore all possible label combinations for the maximum-function. For the sum-product form it is clear that we have to visit each combination once but for the maximum-function we suspect that we can do better.

In the spirit of [8] [19] [20], we may sort the terms by *source variables* x_s and x_t, yielding

$$m_{stuv}(x_u, x_v) = (m_{su} m_{tv})^{-1} \max_{x_s, x_t} (P_{st} M_{su} M_{tv})$$

where we define

$$M_{su} = P_s P_{su}$$
$$M_{tv} = P_t P_{tv}$$

Consider that P_{st}, M_{su} and M_{tv} are all non-negative matrices, while m_{su} and m_{tv} denote non-negative vectors. We observe that without P_{st} we could separate the maximum-function into the product of two maximum-functions:

$$\max_{a,b} (a\, b) = \max_{a} (a) \max_{b} (b)$$

But as the third term P_{st} involves both source variables, this idea is not directly applicable to our message update rule. Nevertheless, we are inspired by the idea of separating the maximum function.

We may assume that the maximum message value is likely to consist of relatively large factors P_{st}, M_{su} and M_{tv}. Thus, the basic idea is to start at the maximum values for M_{su} and M_{tv} and then systematically decrease the factors until the product of both factors and P_{st} is assured to be maximal.

One may be tempted to spare the effort and instead simply ignore P_{st} or take its maximal value into account. However, as the following example indicates, this idea would sacrifice too much accuracy of the GBP algorithm. We shall encounter this example throughout this section to illustrate the fundamental ideas of the algorithm.

Let P_{st} be a $s \times t$ matrix, M_{su} be a $s \times u$ matrix and M_{tv} be a $t \times v$ matrix with values

$$P_{st} = \begin{pmatrix} 1 & 4 \\ 2 & 1 \end{pmatrix}, \quad M_{su} = \begin{pmatrix} 1 & 3 \\ 2 & 2 \end{pmatrix} \quad \text{and} \quad M_{tv} = \begin{pmatrix} 2 & 4 \\ 3 & 1 \end{pmatrix}.$$

Then we can compute the products of these three terms for all possible label combinations of s and t:

$$\begin{aligned} T_{11} &= \begin{pmatrix} 2 & 4 \\ 6 & 12 \end{pmatrix} \cdot 1 &&= \begin{pmatrix} 2 & 4 \\ 6 & 12 \end{pmatrix} \\ T_{12} &= \begin{pmatrix} 3 & 1 \\ 9 & 3 \end{pmatrix} \cdot 4 &&= \begin{pmatrix} 12 & 4 \\ 36 & 12 \end{pmatrix} \\ T_{21} &= \begin{pmatrix} 4 & 8 \\ 4 & 8 \end{pmatrix} \cdot 2 &&= \begin{pmatrix} 8 & 16 \\ 8 & 16 \end{pmatrix} \\ T_{22} &= \begin{pmatrix} 6 & 2 \\ 6 & 2 \end{pmatrix} \cdot 1 &&= \begin{pmatrix} 6 & 2 \\ 6 & 2 \end{pmatrix} \end{aligned}$$

where T_{st} denotes the product for the values s and t.

If we take the component-wise maximum of these products, we obtain the desired value for the corresponding message:

$$\max\left(T_{11}, T_{12}, T_{21}, T_{22}\right) = \begin{pmatrix} 12 & 16 \\ 36 & 16 \end{pmatrix},$$

Ignoring P_{st} would lead to the message value $\begin{pmatrix} 6 & 8 \\ 9 & 12 \end{pmatrix}$, while multiplying it with the maximal component of P_{st} yields $\begin{pmatrix} 24 & 32 \\ 36 & 48 \end{pmatrix}$, which is as well fairly imprecise.

For this reason we inspect the idea of systematically decreasing the maximal values of M_{su} and M_{tv} until the combined product with P_{st} is maximal.

In the first step, we sort M_{su} and M_{tv} column-wise, yielding S_{su} and S_{tv}, and construct index matrices which associate an entry e in S_{su} and S_{tv} with the row position of e in M_{su}, respectively M_{tv}. We denote these index matrices by I_{su}^{SM} and I_{tv}^{SM}. Also, we introduce the converse matrices I_{su}^{MS} and I_{tv}^{MS} that specify at which row a value of M_{su} or M_{tv} is stored in S_{su}, respectively S_{tv}. In our example, M_{su} would lead to the matrices

$$S_{su} = \begin{pmatrix} 2 & 3 \\ 1 & 2 \end{pmatrix}, \quad I_{su}^{SM} = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \quad \text{and} \quad I_{su}^{MS} = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

where we implicit descending order in the columns.

Next, we sort P_{st} and group the indices slightly different to the above method: We sort the whole matrix and convert it into *vector form* which makes I_{su}^{SM} a vector that contains both row and column position. The corresponding index matrix for reaching the values in S from M is a matrix that contains vector indices. For our example, we obtain

$$S_{st} = \begin{pmatrix} 4 & 2 & 1 & 1 \end{pmatrix}^T, \quad I_{st}^{SM} = \begin{pmatrix} (1,2) & (2,1) & (2,2) & (1,1) \end{pmatrix}^T \quad \text{and}$$

$$I_{st}^{MS} = \begin{pmatrix} 3 & 1 \\ 2 & 4 \end{pmatrix}$$

where the ordering of $(1,1)$ and $(2,2)$ is arbitrary.

This completes the construction phase and we can explore label combinations for s and t that are likely to determine the maximal message value. As the columns of I_{su}^{SM} are independent of each other, we have to proceed for each combination of u and v in separation. Theoretically, two possible combinations are suited to start with. On the one hand, we can take the maximal value of M_{su} and M_{tv}, i.e. the entries of the first row in S_{su} and S_{tv}, and multiply it with the corresponding entry of P_{st}. On the other hand, we can begin with the maximal value of P_{st} and compute the product with the corresponding entries in M_{su} and M_{tv}.

We prefer the former alternative, since it is computationally less expensive. The problem of the latter approach is that we can just vaguely estimate how well a computed product relates to other possible products. We have no sorting of the possible products between S_{su} and S_{tv} and computing them would ruin most of the speedup potential.

Thus, for each label of u and v we begin with computing

$$S_{su}(1,u_S) \cdot S_{tv}(1,v_S) \cdot P_{st}\left(I_{su}^{SM}(1,u_S), I_{tv}^{SM}(1,v_S)\right).$$

where u_S denotes the label index of u in S_{su} and v_S the label index of v in S_{tv}. In other words, they refer to the column position in these matrices.

Assuming that we like to evaluate the probability for the first possible label of u and the first label of v, i.e. $u_S = v_S = 1$, we obtain for our example

$$\begin{aligned} S_{su}(1,1) \cdot S_{tv}(1,1) \cdot P_{st}\left(I_{su}^{SM}(1,1), I_{tv}^{SM}(1,1)\right) &= S_{su}(1,1) \cdot S_{tv}(1,1) \cdot P_{st}(2,2) \\ &= 2 \cdot 3 \cdot 1 \\ &= 6 \end{aligned}$$

If $P_{st}(1,1)$ is equal to $S_{st}(1,1)$, we certainly found the maximal value and can terminate. But let us assume that $P_{st}(1,1)$ differs from $S_{st}(1,1)$, as in the example above. Then we can mark the temporary maximum combination in I_{st}^{SM} as being *visited* and *maximal*. The maximum value is stored in $r_{\max}$. For the example this yields:

$$I_{st}^{SM} = \Big((1,2) \quad (2,1) \quad (2,2)^{*\surd} \quad (1,1)\Big)^T$$

where the asterisk denotes that the combination is temporarily maximal and the tick that it has been visited. The new maximum product is

$$r_{\max} = 6.$$

We know that any other combination of s_S and t_S[1] is smaller if its entry in I_{st}^{SM} is to the right of the asterisk position $i_{\max}$. Thus, if the number of *unvisited* combinations i_{left} to the left of $r_{\max}$ is "fairly small", i.e. $i_{\text{left}} \leq \beta$ for some preset bound β, we solely have to check if any of this small number of combinations is bigger than our current maximum. Our algorithm follows this enticing path to the maximum value, but with a minor modification. Only in every second iteration we pick one of the i_{left} combinations; during the intermediate iterations we follow the

[1] In analogy to the definition of u_S and v_S the variables s_S and t_S shall refer to the label index of s and t in S_{su} and S_{st}, i.e. the row positions in these matrices.

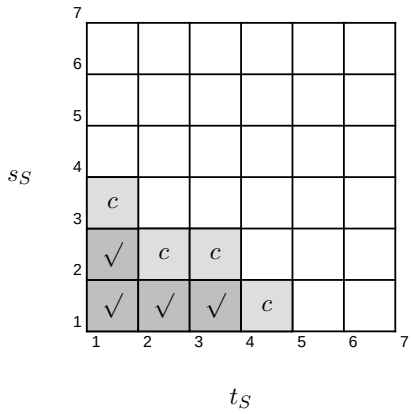


Figure 5.9: A graphical depiction of the candidates for the next combination of s_S and s_T. Visited combinations are marked with a tick on dark gray background. The light gray fields with a c denote possible candidates for the maximal unvisited combination. White fields are unvisited and are not eligible as the next possible combination.

previous scheme of determining s_S and t_S through S_{su} and S_{tv}. The advantage is that we are guaranteed to terminate within $2 \cdot i_{\text{left}}$ iterations but may succeed even earlier when the following situation arises. Assume that the product of the maximal unvisited combination in S_{su} and S_{tv}, multiplied with the maximal unvisited value in S_{st} is not larger than our current maximum $r_{\max}$. Then we can be confident that no other unvisited combination exceeds $r_{\max}$, and are safe to return it as the maximal value. We shall see in a minute where this check is established in our algorithm.

How do we proceed if i_{left} is not larger than β? In this case we investigate the next biggest combination of s_S and t_S in S_{su} and S_{tv}. Two candidates are eligible: Either we multiply the next biggest value in S_{su} with the current selection in S_{tv}, or we multiply the current selection of S_{su} with the next biggest value in S_{tv}. In our example, we evaluate

$$S_{su}(2,1) \cdot S_{tv}(1,1) = 1 \cdot 3 = 3 \quad < \quad S_{su}(1,1) \cdot S_{tv}(2,1) = 2 \cdot 2 = 4.$$

We select the next biggest combination for s_S and t_S, which is $(s_S, t_S) = (1, 2)$ in our example, and multiply it with the largest unvisited value in S_{su} to check if it is larger than our current maximum. Suppose this is fulfilled. Then we calculate the product of our combination and update $r_{\max}$. In addition, we mark the new combination as being visited and update i_{left}. If i_{left} is zero, we can return $r_{\max}$. If not, we have to check if it is smaller than β to possibly change the visiting pattern as explained above. In the residual case, i.e. $i_{\text{left}} \geq \beta$, we proceed for the current combination analogous to the previous one: We search for the next biggest combination of s_S and t_S. Possible candidates are the already computed $(s_S, t_S) = (2, 1)$ and the next two biggest values starting from the previous combination $(s_S, t_S) = (1, 2)$ which are $(s_S, t_S) = (2, 2)$ and $(s_S, t_S) = (1, 3)$. Figure 5.9 helps us visually comprehend the set of possible candidates. All visited combinations are marked with a tick, possible candidates are labeled with *cand*. We notice that all candidate nodes are adjacent to the already visited combinations. Once we have ascertained the new combination of

s_S and t_S, we follow the usual pattern that has been explained for $(s_S, t_S) = (1, 2)$. The algorithm terminates for a specific combination of u_S and v_S if either one of the two checks (against $r_{\max}$ or i_{left}) fails, or if we exhaustively explored all possible combinations of s_S and t_S.

On page 63 we can see the pseudo-code notation of the algorithm. It comprises some additional variables that we omitted so far. In line 10 we detect the first occurrence of the variable *side* which shall indicate the alternating pattern for gaining the next combination for s_S and t_S. If it is zero, we express that I_{su}^{SM} and I_{tv}^{SM} determine the next combination. Set to 1, we reverse the order and obtain the next combination through the smallest unvisited index in I_{st}^{SM}. The variable i_{first} refers to the maximal unvisited entry in S_{st} and as usual k denotes the number of possible labels. Furthermore, we distinguish s_S and t_S from s_M and t_M. The former pair denotes row and column position in S_{su} and S_tv, the latter pair refers to the unsorted matrices M_{su} and M_{tv}. Both pairs are necessary to jump between the index positions of the S-, P- and M-matrices.

In line 53, 54 and 58 we remain informal for not scattering the pseudo-code with implementation details. Nevertheless, these lines deserve some further study. Variable i_{left} shall only be updated to its precise value if we have found a new maximum and infer from S_{su} and S_{tv} to P_{st}. The value is determined by the number of unvisited combinations in I_{st}^{SM} that are left to the current maximum index $i_{\max}$. In all other cases, i.e. we infer from the opposite direction or $i \neq i_{\max}$, we decrease its value by one. We typically keep track of the visited combinations in a *union-find* data structure, e.g. a set, and additionally mark the position of visited combinations in a vector. This ensures that lines 53 and 58 are single instructions and the update of i_{left} is proportional to the number of visited combinations.

The return statement in line 23 is a technical detail to prevent redundant computations of already visited combinations. It is not a check as in line 44 or 55 that allows us to to skip unvisited combinations.

Algorithm 7 Accelerating message update rule for MAP estimate

Require: β is set to a natural number.

Create a column-wise sorted matrix S_{su} out of M_{su} and build two index matrices I_{su}^{SM} and I_{su}^{MS}. Analogously, create S_{tv}, I_{tv}^{SM} and I_{tv}^{MS}.
Create a vector S_{st} which contains all entries of M_{st} in sorted order and construct a corresponding index vector I_{st}^{SM} and an index matrix I_{st}^{MS}.
for all u_S **do**
  for all v_S **do**
    $s_S \Leftarrow 1$ ▷ Init variables
    $t_S \Leftarrow 1$
    $i_{\text{first}} \Leftarrow 1$
    $r_{\max} \Leftarrow -1$
    $i_{\max} \Leftarrow -1$
    $side \Leftarrow 0$
    $i_{\text{left}} \Leftarrow k^2$
    $c \Leftarrow (S_{su}(s_S, u_S) \cdot S_{tv}(t_S, v_S),\ s_S,\ t_S)$
    $cand$.insert(c)
    while $i_{\text{left}} > 0$ **do**
      if $side = 0$ **then**
        repeat
          $c \Leftarrow \max_{c(1)}(cand)$
          $cand$.delete(c)
          $s_M = S_{SU}(c(2), u_S)$
          $t_M = S_{TV}(c(3), v_S)$
        until $((s_M, t_M)$ in I_{st}^{SM} is *unvisited*$)$ or $(cand$ is *empty*$)$
        if cand is *empty* **then**
          return $r_{\max}$
        end if
        $prod \Leftarrow c(1)$
        $s_S \Leftarrow c(2)$
        $t_S \Leftarrow c(3)$
      else if $side = 1$ **then**
        $s_S = I_{su}^{MS}\left(I_{st}\left(i_{\text{first}}\right)(1), u_S\right)$
        $t_S = I_{tv}^{MS}\left(I_{st}\left(i_{\text{first}}\right)(2), v_S\right)$
        $prod \Leftarrow S_{su}(u_S, u_S) \cdot S_{tv}(t_S, v_S)$
        $s_M \Leftarrow I_{su}^{SM}(s_S, u_S)$
        $t_M \Leftarrow I_{tv}^{SM}(t_S, v_S)$
      end if
      if $s_S < k$ **then**
        $c \Leftarrow (S_{su}(s_S + 1, u_S) \cdot S_{tv}(t_S, v_S),\ s_S + 1,\ t_S)$
        $cand$.insert(c)
      end if
      if $t_S < k$ **then**
        $c \Leftarrow (S_{su}(s_S, u_S) \cdot S_{tv}(t_S + 1, v_S),\ s_S,\ t_S + 1)$
        $cand$.insert(c)
      end if

```
43:             if prod · S_st(i_first) ≤ r_max then
44:                 return r_max                                  ▷ Check r_max
45:             end if
46:             r ⇐ prod · P_st(s_M, t_M)
47:             i ⇐ I_st^MS(s_M, t_M)
48:             if r > r_max then
49:                 r_max ⇐ r
50:                 i_max ⇐ i
51:             end if
52:             if i ≤ i_max then
53:                 Mark (s_M, t_M) in I_st^SM as visited.
54:                 Update i_left.
55:                 if i_left = 0 then return r_max               ▷ Check i_left
56:                 end if
57:             end if
58:             Update i_first
59:             if i_left < β then
60:                 side ⇐ 1
61:             else
62:                 side ⇐ 0
63:             end if
64:         end while
65:     end for
66: end for
```

Let us simulate the algorithm for $u_S = v_S = 1$ and $\beta = 2$, while confining ourselves to the essential steps. The notation is rather brief, but hopefully self-explanatory.

- $s_S = 1,\ t_S = 1$:
  - $s_M = I_{su}^{SM}(1,1)$ and $t_M = I_{tv}^{SM}(1,1)$
  - Evaluate $S_{su}(1,1) \cdot S_{tv}(1,1) \cdot P_{st}(s_M, t_M) = 2 \cdot 3 \cdot 1 = 6$
    $\Rightarrow r_{\max} = 6,\ I_{st}^{SM} = \Big((1,2) \quad (2,1) \quad (2,2)^{*\surd} \quad (1,1)\Big)^T,\ i_{\text{left}} = 2,\ i_{\text{first}} = 1$
  - Check $i_{\text{left}} < \beta \Rightarrow$ False
  - Add candidates:
    $(s_S, t_S) = (2,1)$ with value $S_{su}(2,1) \cdot S_{tv}(1,1) = 3$ and
    $(s_S, t_S) = (1,2)$ with value $S_{su}(1,1) \cdot S_{tv}(2,1) = 4$
    $\Rightarrow$ Next combination is $(s_S, t_S) = (1,2)$.
- $s_S = 1,\ t_S = 2$:
  - $s_M = I_{su}^{SM}(1,1)$ and $t_M = I_{tv}^{SM}(2,1)$
  - Check $S_{su}(1,1) \cdot S_{tv}(2,1) \cdot S_{st}(1) = 4 \cdot 4 = 16 > r_{\max} \Rightarrow$ True

- Evaluate $S_{su}(1,1) \cdot S_{tv}(2,1) \cdot P_{st}\left(I_{su}^{SM}(1,1), I_{tv}^{SM}(2,1)\right) = 4 \cdot 2 = 8$
  $\Rightarrow r_{\max} = 8,\ I_{st}^{SM} = \Big((1,2) \quad (2,1)^{*\surd} \quad (2,2)^{*} \quad (1,1)\Big)^T,\ i_{\text{left}} = 1,\ i_{\text{first}} = 1$
- Check $i_{\text{left}} < \beta \Rightarrow$ True $\Rightarrow$ Alternating visiting scheme. Next (s,t) is determined by maximal unvisited entries in S_{su} and S_{tv}.
- Check $i_{\text{left}} = 0 \Rightarrow$ False
- Add candidates:
  $(s_S, t_S) = (2,2)$ with value $S_{su}(2,1) \cdot S_{tv}(2,1) = 2$
  $\Rightarrow$ Next combination is $(s_S, t_S) = (2,1)$.

- $s_S = 2,\ t_S = 1$:
  - $s_M = I_{su}^{SM}(2,1)$ and $t_M = I_{tv}^{SM}(1,1)$
  - Check $S_{su}(2,1) \cdot S_{tv}(1,1) \cdot S_{st}(1) = 3 \cdot 4 = 12 > r_{\max} \Rightarrow$ True
  - Evaluate $S_{su}(2,1) \cdot S_{tv}(1,1) \cdot P_{st}(s_M, t_M) = 3 \cdot 4 = 12$
    $\Rightarrow r_{\max} = 12,\ I_{st}^{SM} = \Big((1,2)^{*\surd} \quad (2,1)^{*} \quad (2,2)^{*} \quad (1,1)\Big)^T,\ i_{\text{left}} = 0$
  - Check $i_{\text{left}} < \beta \Rightarrow$ True $\Rightarrow$ Alternating visiting scheme. Next (s,t) is determined by maximal unvisited entry in S_{st}.
  - Check $i_{\text{left}} = 0 \Rightarrow$ True $\Rightarrow$ Return $r_{\max} = 12$.

We may skip the evaluation of the residual combinations of u and v. To get an impression of the speedup we highlight all visited combinations of s and t in T_{uv}. Note that we label combinations additionally with $_<$ if they did not pass the maximum check in line 19 of the algorithm, whereas we give the flag $_0$ to all combinations for which i_{left} evaluates to 0.

$$
\begin{aligned}
T_{11} &= \begin{pmatrix} 2 & 4 \\ 6 & \mathbf{12} \end{pmatrix} \\
T_{12} &= \begin{pmatrix} \mathbf{12}_0 & \mathbf{4}_< \\ \mathbf{36}_0 & \mathbf{12}_< \end{pmatrix} \\
T_{21} &= \begin{pmatrix} \mathbf{8} & \mathbf{16} \\ 8 & \mathbf{16} \end{pmatrix} \\
T_{22} &= \begin{pmatrix} \mathbf{6} & 2 \\ 6 & 2 \end{pmatrix}
\end{aligned}
$$

Thus, the algorithm visits only 9 out of 16 combinations. We may argue that the extra operations of the algorithm clearly predominate the speedup. However, these extra cost are (nearly) constant in the number of visited operations and for any $k > 6$ the algorithm already seems to be worthwhile. Experimental results show that we can often compute the best combination by inspecting a small number α of combinations. The algorithm demonstrates complexity of k^3 in the number of labels k, in contrast to the standard algorithm which runs in k^4. The reason is that the factors S_{su} and

S_{tv} are heavily influenced by P_s and P_t respectively. As a reminder, we defined P_s for cluster messages in 2D by

$$P_s = \phi_s m_{as} m_{cs}.$$

Thus, P_s consists of one data potential function ϕ_s and two edge messages m_{as} and m_{cs} which inform node s about its most likely probability values from the outward perspective. At least the two messages often attribute relatively high probabilities to a very small number of labels, whereas the majority of labels is considered to be much less likely. This means for P_s and P_t that they contain a small number of large values which raises the likelihood that any of these values emerges in the optimal combination of s and t.

A nice property of the algorithm is that it easily extends to our three-dimensional grid graphs. Recall that we also expressed them with the interim products P_s, P_t, P_{su}, P_{sv} and P_{st} which means that we can apply the algorithm without any change. Also, the principal idea of the algorithm should translate to other Markov random fields, although its outline may become more intricate and the speedup may vary significantly.

Chapter 6

Experimental Results

6.1 Overview

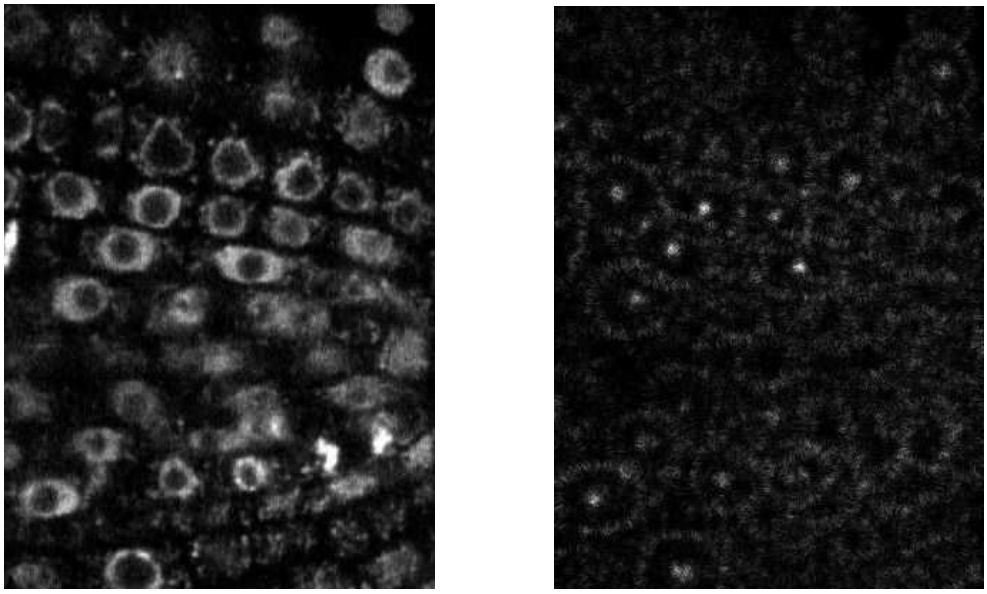
In this chapter we evaluate the presented acceleration techniques on an image segmentation problem from biology. We shall demonstrate the potential of generalized belief propagation under realistic conditions but we also expose the attractiveness of Markov random field approaches to labeling problems. We begin with a brief description of the image segmentation problem and explain how we set the data and the smoothness potential functions in the corresponding Markov random field. Afterwards, we quantify the performance of our acceleration techniques on this real-world example.

6.2 Image segmentation problem

As depicted in figure 6.1(a), we are given a cross section of a three-dimensional tissue probe that has been recorded by a confocal laser scanning microscope (LSM) [6]. Our task is to segment the midpoints of the cells to visualize their distribution, which might be interesting for tracking and analyzing relative cell movements over time.

In a fist step, we extract features with a *Hough transform* (cf. section 2.1.2) for exploiting the characteristic shape of the cells. For differing radii and gradient direction we obtain a set of features that we superimpose to better contrast possible cell midpoints. Figure shows the result of summing the features from the Hough transform where white dots indicate possible midpoints. We observe the challenge of the data: It is degraded by noise and some point clouds correspond to cell mid points whereas others lie in between. Many point-based, edge-based and region-based approaches from the literature struggle on this problem because of the noisy data and the lack of obvious local constraints for classifying the point candidates. Markov random field approaches, though, offer an elegant representation for modeling the affinity to the data and for encoding *global* a priori information. How do we model the parameters for this example?

A straightforward approach is to decompose the problem into two subsequent inference passes. In the first pass we aim at combining point clouds to coherent regions and at "denoising" the image from the residual point candidates. For both the data potential and the smoothness potential we use a *piecewise constant prior*.



(a) Original image (b) After Hough transform 6.2

Figure 6.1: On the left we see the cross section of a tissue probe; on the right we see the summed Hough transform features of the left image.

The *data potential* of image node x_s is modeled by

$$V_s(x_s) = \begin{cases} 0 & g_s < \tau/2 \\ \tau & \text{else} \end{cases}$$

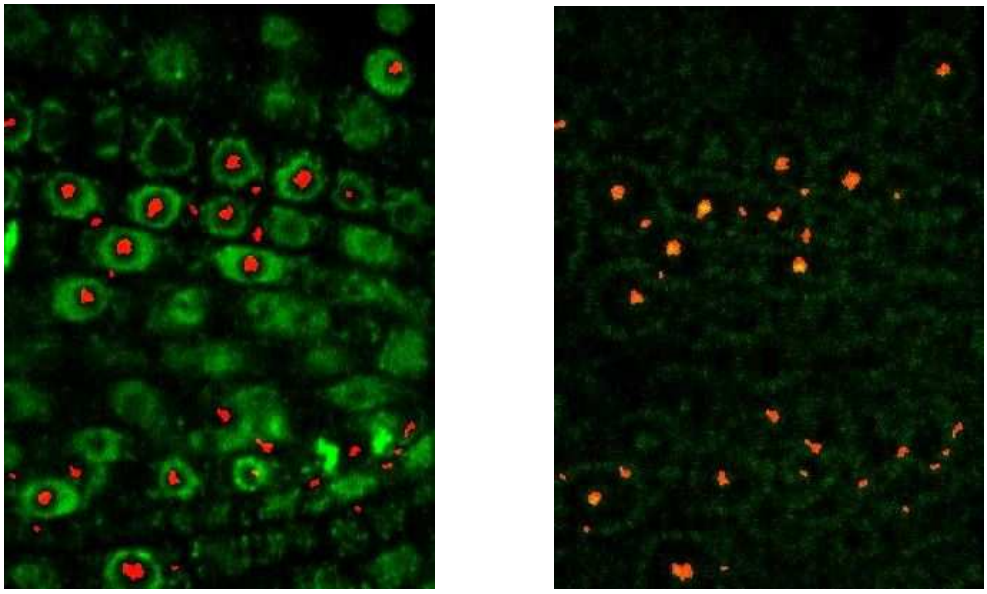
where the hidden image node $x_s = (l_s, g_s)$ and the observed image node $y_s = g_s$. The parameter τ defines a threshold that we set to 90 for this example. For the *smoothness potential* we use an *Ising model*. With these parameters we perform the generalized belief propagation algorithm on the feature image. We use a first order neighborhood system and restrict ourselves to cluster regions of size four. Figure 6.2 shows the result.

In the second pass, we process the intermediate image by defining a Markov random field on a higher abstraction level. Instead of using pixels in a grid graph, we model a general Markov random field that consists of coherent point regions as nodes. Although our acceleration techniques are designed for grid graphs and can thus not be applied, we shall complete the modeling of this example to illustrate the potential of Markov random fields for labeling problems.

The *data potential* of the second pass is expressed as a *truncated linear model*

$$V_s(x_s) = \min(a_s, \tau)$$

where $x_s = (l_s, a_s)$ and $y_s = a_s$. The image feature a_s shall denote the *area* of an object region in pixels. The parameter τ is set to 255, the maximal value for 8 bit gray level. It is motivated by the simple observation that large object regions are more likely to express a midpoint of a cell than small regions.



(a) Result from first pass against original

(b) Result from first pass against image after Hough transform

Figure 6.2: The result of the first pass is depicted on red. It is contrasted against the original image and the Hough transform. Note that several point regions are erroneously situated between cells.

The *smoothness potential* cannot be attributed to any of the three prior categories. Depending on the prior information we shall assign one of the following three potential functions to it:

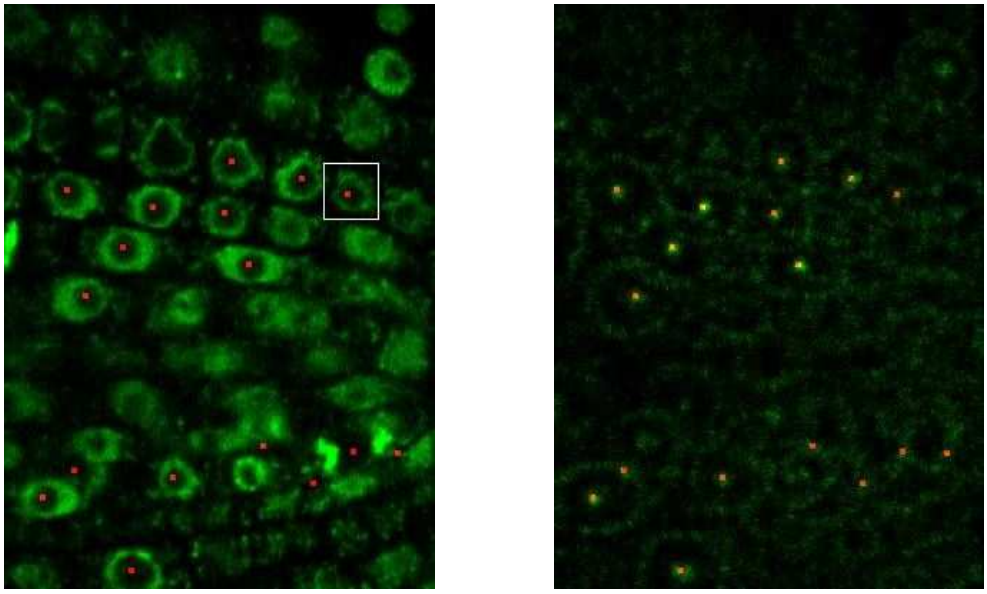
$$V_{st}^{inc}(x_s, x_t) = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$V_{st}^{com}(x_s, x_t) = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

$$V_{st}^{neut}(x_s, x_t) = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

They reflect an *incompatible*, *compatible*, and a *neutral* smoothness potential.

Two local decision rules are used for classifying the candidate midpoints: For the first rule we measure the distance between the midpoints of two object regions. If the distance is below a certain threshold, which we set to 35 for our example, we consider the two nodes as being *incompatible*. Otherwise, we apply a second rule. According to figure 6.1(a) a distinctive feature of the original image data is the quite regular distribution of the cells. Adjacent midpoints are oriented similar relative to their closest neighbors. For this reason we evaluate for all neighbor nodes x_s and x_t the dot product between all combinations of vectors n_s and n_t, where n_s denotes a vector between x_s to any of its four neighbors and n_t stands for a vector between x_t and any its four neighbors. If $\arccos(\langle n_s, n_t \rangle) < \alpha$ for any combination of n_s and n_t, we regard the two nodes x_s and x_t as *compatible*. The threshold α is set to $\frac{5}{360}2\pi = 0.0873$ in our implementation. In all other cases the node pair is classified as *neutral*.



(a) Result from second pass against original

(b) Result from second pass against image after Hough transform

Figure 6.3: The final result of this image segmentation problem is depicted. On the left we contrast the result of the second pass against the original image; on the right we highlight the result against the feature image from the Hough transform.

If we apply generalized belief propagation to the second Markov random field, we obtain the images depicted in figure 6.3. Note that for instance the midpoint surrounded by a white square has small data support but it is preserved due to its geometric constellation with large data points. Of course, the result is not optimal as we lose some correct midpoints from figure 6.2. We might therefore think of a third constraint that rewards isloated data points with high confidence levels. Alternatively, we can improve on the parameter values. They have been set in an adhoc fashion and certainly not optimal. However, we can summarize that Markov random fields offer a powerful tool to solve labeling problems.

6.3 Results

In this thesis, we presented four different acceleration techniques: (1) *hierarchical initialization*, (2) *active message technique*, (3) *caching and multiplication* and (4) the *acceleration for the MAP estimate*. The first technique has been discovered by [8] who formulated the approach for the standard belief propagation algorithm. Out of his three acceleration techniques the multi-grid BP approach shows the most impressive results. We have adopted the technique for the generalized belief propagation algorithm by presenting to detailed propagation update scheme. However, we have not encountered comparable speedups as [8]. One reason might be that we neglected the smoothness potential design for each hierarchy level. We shall pursue this approach in the future though.

The second technique called *active message technique* affects the running time of

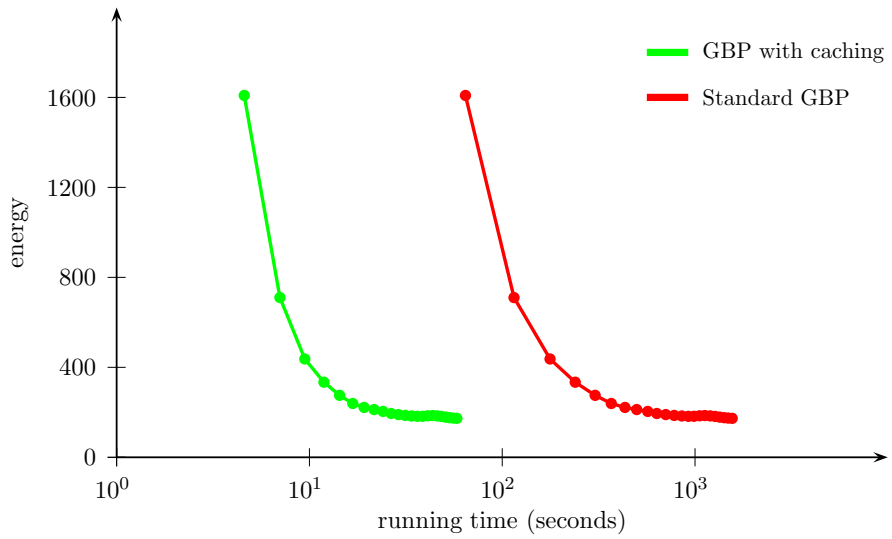


Figure 6.4: Energy of the image segmentation problem as a function of the running time. Each dot stands for a completed iteration of the inference algorithm. Note the logarithmic scale of the abscissa.

our algorithm unpredictably. For the above image segmentation problem we have not gained any speedup whereas for other problem that consist of more homogeneous regions we could approximately halve the running time. The challenge is to formulate a general heuristic that balances accuracy and speed.

The *caching and multiplication* technique leads to tremendous speedups as figure 6.4 illustrates. We analyzed the energy of the underlying graphical model over time and the result is striking. Our technique clearly improves on the standard implementation.

The forth technique *acceleration of the MAP estimate* similarly decreases the execution time. In figure 6.5 we depict the running time as a function of the number of labels k. As our biological segmentation problem differentiates solely two labels, we were forced to evaluate the significance of this technique on another problem set. We chose to compare the standard matrix multiplication[1] with our proposed multiplication scheme by measuring the running time for multiplying random potential functions of size $k \times k$. For each value of k we computed 100 random smoothness potentials and averaged the execution time. Thus, not only do our results reflect the speed but also the flexibility of our technique. Figure 6.6 shows the ratio of the running times for the standard generalized belief propagation and our acceleration technique. We observe a linear dependence.

[1] We used the `blitz++` library

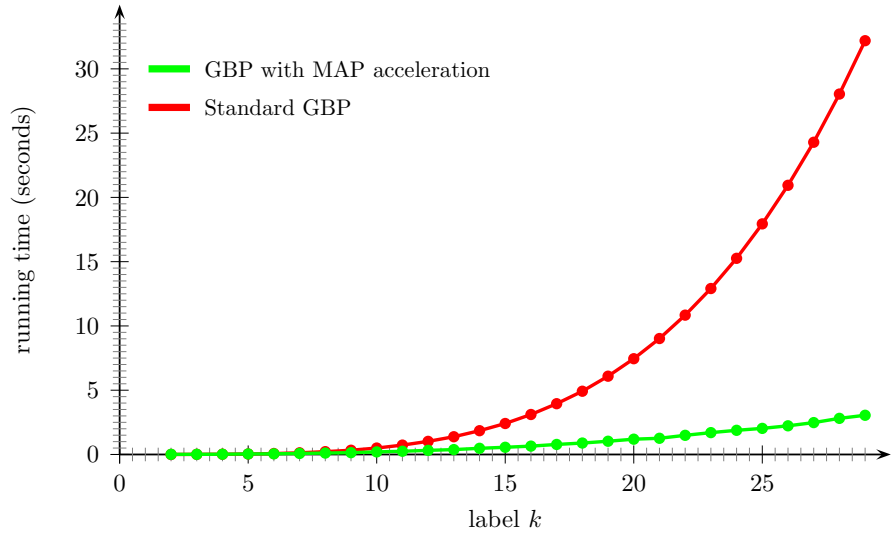


Figure 6.5: Running time as a function of the number of labels k.

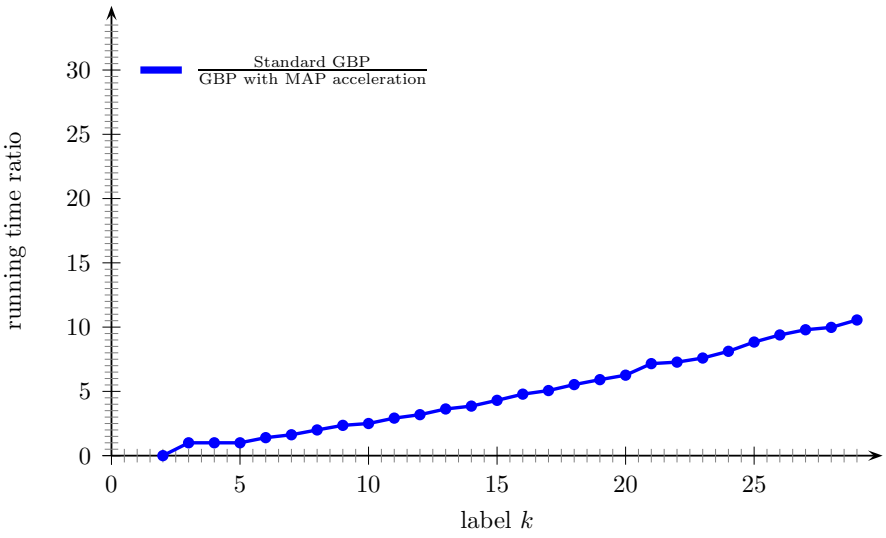


Figure 6.6: Ratio of standard GBP to GBP with MAP acceleration as a function of the number of labels k.

Chapter 7

Conclusion

In this thesis, we concentrated on developing acceleration techniques for the generalized belief propagation algorithm of [39]. The accuracy, flexibility and convergence behavior of this technique have been proven in several experiments. Also, the generalized belief propagation is theoretically justified by approximating the Kikuchi free energy. However, its execution speed has been a severe shortcoming.

For this reason we introduced four approaches to accelerate the algorithm. Two of them, the *caching and multiplication* technique and the *acceleration for the MAP estimate*, performed surprisingly well in our experiments. The other two techniques have to be further developed in the future. For some labeling problems they lead to speed-ups but a general reliable prediction is not yet possible. For the *hierarchical initialization* technique we already formulated a promising message propagation scheme but we neglected the design of potential function throughout the hierarchy. The *active message technique* lacks a sophisticated heuristic that encompasses several eventualities when reducing message from active to passive. With some further testing we are confident, though, to succeed with this technique.

Apart from acceleration techniques we also covered *image segmentation* problems. In the previous chapter we exemplified how Markov random fields can be easily adopted to capture global constraints. The results are already encouraging although we have not invested much effort in deriving highly specialized potentials. In combination with learning algorithms and the consideration of three-dimensional constraints we are confident to obtain satisfactory solutions to a set of similar labeling problems.

Notation

S	The set of sites or locations
$s \in S$	Site
g_s	Intensity value at site s
g	Pattern of intensities
l_s	Label at site s
l	Label configuration, labeling
x_s	Hidden image value at site s
y_s	Observed image value at site s
X_s	Random variable
$\mathbf{X}_s$	State space
$\mathbf{X} = \prod_{s \in S} \mathbf{X}_s$	Configuration space of sites $s \in S$
x	Hidden image
y	Observed image
k	Number of possible labels (at any site s)
n	Number of sites in S
∂	Neighbourhood system
$s \sim t$	s and t are neighbours
$\mathcal{C}$	The set of cliques
$C \in \mathcal{C}$	Clique
ψ_C	Potential function of clique C
$E(x)$	Energy function
$E_{\text{data}}(x)$	Data energy
$E_{\text{prior}}(x) = E_{\text{smooth}}(x)$	Prior (smoothness) energy
$U_s^w(x_s)$	Node (data) potential at site s in x with parameter w
$U_{st}^w(x_s, x_t)$	Edge (prior, smoothness) potential at sites s and t in x with parameter w
$V_s(x_s)$	Node (data) potential (if w is linear for U_s)
$V_{st}(x_s, x_t)$	Edge (prior, smoothness) potential (if w is linear for U_{st})
$\phi_s(x_s)$	Data potential function
$\psi_{st}(x_s, x_t)$	Prior (smoothness) potential function

$\mathcal{B}^i$	Set of blocks
b_k^i	The k-th block at scale i
x_k^i	The hidden image value of block k at scale i
$b_s(x_s)$	Belief at site s of the hidden image x
$m_{su}(x_u)$	Edge message from site s to site u in x
$m_{stuv}(x_u, x_v)$	Cluster message from the sites s, t to the sites u, v in x
P_s	Interim product at site s
P_{st}	Interim product at sites s and t

List of Figures

3.1 Three kinds of graphical models (from [11]) 13
3.2 Factor graph with pairwise potentials (from [11]) 15
3.3 Problems with undirected graphs and factor graphs (from [11]) 15
3.4 Expressive power of directed and undirected graphs (from [11]) 16
3.5 Typical neighborhood systems for a Markov random field. (a) and (b) show first- and second-order neighborhood system in 2D. (b) and (c) depict the same for 3D. 18
3.6 Graph of $U_{st}(x_s, x_t)$ for a everywhere smooth prior 21
3.7 Graphs of $U_{st}(x_s, x_t)$ for two piecewise constant priors. 21
3.8 Graphs of $U_{st}(x_s, x_t)$ for two piecewise smooth priors 22

4.1 A Markov random field with pairwise potential functions. 31
4.2 Diagrammatic representations of the belief formula for single-node and two-node beliefs. The arrows indicate messages that affect the corresponding hidden nodes. Undirected lines denote potential functions (from [40]). 32
4.3 A diagrammatic representation of the message update rule $m_{ts}(x_s)$. The summatoin symbol indicates that we sum over all possible labels for node t (from [40]). 33
4.4 A directed chain graph for four nodes. 33
4.5 On the left we see a diagrammatic representation of the basic clusters; on the right we see a possible region graph for the corresponding Markov random field. 37
4.6 On the left we see a diagrammatic representation for a single-node belief region, highlighted with a gray background. Its formula is $b_5 = k\,\phi_5\, m_{25}m_{45}m_{65}m_{85}$. The right figure depicts the corresponding region graph. Blue directed edges denote messages between single nodes. . . 39
4.7 On the left we see a diagrammatic representation for a two-node belief region. Its formula is $b_{45} = k\,\phi_4\phi_5\psi_{45}\, m_{25}m_{65}m_{85}m_{1245}m_{7845}$. The right figure depicts the corresponding region graph. Green directed edges denote messages between edges, i.e. two-node beliefs. 40
4.8 On the left we see a diagrammatic representation for a four-node belief region. Its formula is $b_{1245} = k\,\phi_1\phi_2\phi_4\phi_5\psi_{14}\psi_{25}\psi_{45}\, m_{65}m_{85}m_{3625}m_{7845}$. The right figure depicts the corresponding region graph. 40

4.9 In the upper figure we see a diagrammatic representation of the GBP message update rule for a cluster message. The source belief region of the message is depicted in light gray, whereas dark gray indicates the target belief region. The red directed edge denotes the cluster message m_{4512} which leads from edge 45 to edge 12. Its formula is $m_{4512} = \phi_1\phi_2\phi_4\phi_5\psi_{12}\psi_{14}\psi_{25}\psi_{45}\, m_{74}m_{65}m_{85}m_{3645}m_{7845}$. Below it we have depicted the complete region graph as it is commonly constructed in the GBP algorithm. 41

5.1 Diagrammatic representation of the first two hierarchy levels. The highlighted regions indicate the nodes on the respective level and simultaneously blocks b_k that comprise the contained nodes of level 0. The bold edges are updated before the normal edges during the propagation update scheme. 47

5.2 A diagrammatic representation of the hierarchical initialization in a three-dimensional grid graph. The gray regions correspond to blocks in the observation field, respectively nodes on the next coarser level. Bold edges indicate messages that are evaluated before normal edge messages. The green cluster message is an example for the type of cluster message that is preferably evaluated. Second are messages that are bordered by two bold lines, depicted in yellow. Finally, we calculate cluster messages within blocks, such as the red region. 50

5.3 A diagrammatic representation of the messages that influence the single-node belief at site s in a two-dimensional grid. 52

5.4 A diagrammatic representation of all edge messages (in red) that are contained in the same two-node belief region $R = \{s, u\}$. Note that the cluster messages from edges to edges are identical in both figures. 52

5.5 A diagrammatic representation of all cluster messages (in red) that are contained in the same four-node belief region $R = \{s, t, u, v\}$. Blue edges stand for edge messages in the nominator, whereas dashed edges are those in the denominator of the corresponding message update rule. Green messages denote as usual cluster messages that influence the value of the red cluster message. We can observe that messages appear within several figures. 53

5.6 A diagrammatic representation of the messages that influence the single-node belief at site s in a three dimensional grid. 56

5.7 A diagrammatic representation of the messages that influence the edge message from site s to site u. 56

5.8 A diagrammatic representation of all messages that influence the red cluster message from edge st to edge uv. Edge messages in the nominator of the message update rule are depicted in blue. If they are in denominator they are dashed black. Green arrows refer to incoming cluster messages. 57

5.9 A graphical depiction of the candidates for the next combination of s_S and s_T. Visited combinations are marked with a tick on dark gray background. The light gray fields with a c denote possible candidates for the maximal unvisited combination. White fields are unvisited and are not eligible as the next possible combination. 61

6.1 On the left we see the cross section of a tissue probe; on the right we see the summed Hough transform features of the left image. 68
6.2 The result of the first pass is depicted on red. It is contrasted against the original image and the Hough transform. Note that several point regions are erroneously situated between cells. 69
6.3 The final result of this image segmentation problem is depicted. On the left we contrast the result of the second pass against the original image; on the right we highlight the result against the feature image from the Hough transform. 70
6.4 Energy of the image segmentation problem as a function of the running time. Each dot stands for a completed iteration of the inference algorithm. Note the logarithmic scale of the abscissa. 71
6.5 Running time as a function of the number of labels k. 72
6.6 Ratio of standard GBP to GBP with MAP acceleration as a function of the number of labels k. 72

Bibliography

[1] Harrison H. Barrett and Kyle Myers. *Foundations of Image Science*. John Wiley & Sons, 2003.

[2] Charles A. Bouman and Michael Shapiro. A multiscale random field model for bayesian image segmentation. *IP*, 3(2):162–177, March 1994.

[3] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(11):1222–1239, 2001.

[4] Robert G. Cowell, Steffen L. Lauritzen, A. Philip David, David J. Spiegelhalter, and David J. Spiegelhater. *Probabilistic Networks and Expert Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[5] Peter Elias, Amiel Feinstein, and Claude E. Shannon. Note on maximum flow through a network. *IRE Transactions on Information Theory IT-2*, pages 117–199, 1956.

[6] Janis Fehr, Olaf Ronneberger, Haymo Kurz, and Hans Burkhardt. Self-learning segmentation and classification of cell-nuclei in 3d volumetric data using voxel-wise gray scale invariants. In Walter G. Kropatsch, Robert Sablatnig, and Allan Hanbury, editors, *DAGM-Symposium*, volume 3663 of *Lecture Notes in Computer Science*, pages 377–384. Springer, 2005.

[7] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Distance transforms of sampled functions. Technical report, Cornell Computing and Information Science, September 2004.

[8] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient belief propagation for early vision. *Int. J. Comput. Vision*, 70(1):41–54, 2006.

[9] Lester R. Ford and Delbert R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, New Jersey, 1962.

[10] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, November 1984.

[11] Zoubin Ghahramani. Lecture 1: Introduction to graphical models. Presentation at Machine Learning Summer School, August 2007.

[12] Basilis Gidas. A renormalization group approach to image processing problems. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(2):164–180, 1989.

[13] Rafael C. Gonzales and Richard E. Woods. *Digital Image Processing.* Prentice Hall, New Jersey, 1993.

[14] D. M. Greig, B. T. Porteous, and Allan H. Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society*, 51(2):271–279, 1989.

[15] Tom Heskes, Kees Albers, and Bert Kappen. Approximate inference and constrained optimization. In *UAI*, pages 313–320, 2003.

[16] Ming Jiang. *Digital Image Processing*, School of Mathematics, Peking University, http://iria.pku.edu.cn/∼jiangm/courses/dip/html/dip.html (checked on 8.11.2007).

[17] Zoltan Kato, Marc Berthod, and Josiane Zerubia. A hierarchical markov random field model and multitemperature annealing for parallel image classification. *CVGIP: Graphical Model and Image Processing*, 58(1):18–37, 1996.

[18] Pushmeet Kohli and Philip H. S. Torr. Effciently solving dynamic markov random fields using graph cuts. In *ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision*, pages 922–929, Washington, DC, USA, 2005. IEEE Computer Society.

[19] M. Pawan Kumar and Philip H. S. Torr. *Fast Memory-Efficient Generalized Belief Propagation*, volume 3954/2006 of *Lecture Notes in Computer Science*, pages 451–463. Springer Berlin / Heidelberg, 2006.

[20] M. Pawan Kumar, Philip H. S. Torr, and Andrew Zisserman. Obj cut. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 18–25 vol. 1, 2005.

[21] Victor S. Lempitsky, Carsten Rother, and Andrew Blake. Logcut - efficient graph cut optimization for markov random fields. In *IEEE International Conference on Computer Vision (ICCV)*, October 2007.

[22] Diane Lingrand and Johan Montagnat. Levelset and b-spline deformable model techniques for image segmentation: A pragmatic comparative study. In *SCIA*, pages 25–34, 2005.

[23] David J. C. Mackay. *Information Theory, Inference & Learning Algorithms.* Cambridge University Press, June 2002.

[24] Kevin Murphy, Yair Weiss, and Michael Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 467–47, San Francisco, CA, 1999. Morgan Kaufmann.

[25] Patrick Pérez. Markov random fields and images. *CWI Quarterly*, 11(4):413–437, 1998.

[26] Linda G. Shapiro and George C. Stockman. *Computer Vision*. Prentice Hall, New Jersey, March 2001.

[27] Noam Shental, Assaf Zomet, Tomer Hertz, and Yair Weiss. Learning and inferring image segmentations using the gbp typical cut algorithm. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, page 1243, Washington, DC, USA, 2003. IEEE Computer Society.

[28] Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Vladimir Kolmogorov, Aseem Agarwala, Marshall F. Tappen, and Carsten Rother. A comparative study of energy minimization methods for markov random fields. In Ales Leonardis, Horst Bischof, and Axel Pinz, editors, *ECCV (2)*, volume 3952 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2006.

[29] Marshall F. Tappen and William T. Freeman. Comparison of graph cuts with belief propagation for stereo, using identical mrf parameters. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, page 900, Washington, DC, USA, 2003. IEEE Computer Society.

[30] Yee Whye Teh and Max Welling. Passing and bouncing messages for generalized inference. Technical Report GCNU TR 2001-01, Gatsby Computational Neuroscience Unit, University College London, 2001.

[31] Yee Whye Teh and Max Welling. The unified propagation and scaling algorithm. In *Advances in Neural Information Processing Systems*, volume 14, 2002.

[32] Olga Veksler. *Efficient graph-based energy minimization methods in computer vision*. PhD thesis, Cornell University, August 1999.

[33] Max Welling. On the choice of regions for generalized belief propagation. In *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence (UAI-04)*, page 585, Arlington, Virginia, 2004. AUAI Press.

[34] Alan S. Willsky. Multiresolution markov models for signal and image processing, 2002.

[35] Gerhard Winkler. *Image Analysis, Random Fields and Markov Chain Monte Carlo Methods*. Springer, February 2006.

[36] Oliver Wirjadi. Survey of 3d image segmentation methods. Technical Report 123, Fraunhofer Institut Techno- und Wirtschaftsmathematik, 2007.

[37] Jonathan S. Yedida, William T. Freeman, and Yair Weiss. Bethe free energy, kikuchi approximations and belief propagation algorithms. Technical Report TR-2001-16, Mitsubishi Electric Research Laboratories, May 2001.

[38] Jonathan S. Yedida, William T. Freeman, and Yair Weiss. Constructing free energy approximations and generalized belief propagation algorithms. Technical Report TR-2004-40, Mitsubishi Electric Research Laboratories, December 2004.

[39] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Generalized belief propagation. In *NIPS*, pages 689–695, June 2000.

[40] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Understanding belief propagation and its generalizations. Technical Report TR-2001-22, Mitsubishi Electric Research Laboratories, January 2001.

[41] Alan L. Yuille. Cccp algorithms to minimize the bethe and kikuchi free energies: convergent alternatives to belief propagation. *Neural Comput.*, 14(7):1691–1722, 2002.