Factor Graphs for Computer Vision and Image Processing

Lawrence Mutimbu

A thesis submitted for the degree of Doctor of Philosophy
The Australian National University

August 2016
Except where otherwise indicated, this thesis is my own original work.

Lawrence Mutimbu
August 30, 2016
Acknowledgments

This work would not have been possible but for my supervisor, Dr. Antonio Robles-Kelly. He has provided me with the direction and guidance that has allowed me to complete my research. Many thanks to the National ICT Australia (NICTA) for funding this research, its staff and fellow students for making the past three and a half years a journey worth to remember. My deepest gratitude also to the ANU for providing the facilities necessary for me to do my research to the best of my ability. I would also like to extend my gratitude to my friends and family; particularly my parents, who have given me the freedom to pursue my dreams.
Abstract

Factor graphs have been used extensively in the decoding of error correcting codes such as turbo codes, and in signal processing. However, while computer vision and pattern recognition are awash with graphical model usage, it is somewhat surprising that factor graphs are still somewhat under-researched in these communities. This is surprising because factor graphs naturally generalise both Markov random fields and Bayesian networks. Moreover, they are useful in modelling relationships between variables that are not necessarily probabilistic and allow for efficient marginalisation via a sum-product of probabilities.

In this thesis, we present and illustrate the utility of factor graphs in the vision community through some of the field’s popular problems. The thesis does so with a particular focus on maximum a posteriori (MAP) inference in graphical structures with layers. To this end, we are able to break-down complex problems into factored representations and more computationally realisable constructions. Firstly, we present a sum-product framework that uses the explicit factorisation in local subgraphs from the partitioned factor graph of a layered structure to perform inference. This provides an efficient method to perform inference since exact inference is attainable in the resulting local subtrees. Secondly, we extend this framework to the entire graphical structure without partitioning, and discuss preliminary ways to combine outputs from a multilevel construction. Lastly, we further our endeavour to combine evidence from different methods through a simplicial spanning tree reparameterisation of the factor graph in a way that ensures consistency, to produce an ensembled and improved result. Throughout the thesis, the underlying feature we make use of is to enforce adjacency constraints using Delaunay triangulations computed by adding points dynamically, or using a convex hull algorithm. The adjacency relationships from Delaunay triangulations aid the factor graph approaches in this thesis to be both efficient and competitive for computer vision tasks. This is because of the low treewidth they provide in local subgraphs, as well as the reparameterised interpretation of the graph they form through the spanning tree of simplexes. While exact inference
is known to be intractable for junction trees obtained from the loopy graphs in computer vision, in this thesis we are able to effect exact inference on our spanning tree of simplexes. More importantly, the approaches presented here are not restricted to the computer vision and image processing fields, but are extendable to more general applications that involve distributed computations.
Contents

Acknowledgments v

Abstract vii

1 Introduction 1
  1.1 Motivation .............................................................. 1
  1.2 Outline ................................................................. 2

2 Literature Review 5
  2.1 Background ............................................................... 5
  2.2 Inference in Graphical Models ......................................... 8
    2.2.1 Inference in MRFs .................................................. 10
    2.2.2 Inference in Factor Graphs ...................................... 18
  2.3 Graphs with Multi-level Interactions .................................. 31
  2.4 Relation to Neural Networks .......................................... 34
  2.5 Applications ............................................................ 36
    2.5.1 Decoding .............................................................. 37
    2.5.2 Wireless Networks .................................................. 39
    2.5.3 Biological Systems ................................................ 40
    2.5.4 Other ................................................................. 41

3 Efficient Inference in Factor Graphs 43
  3.1 Introduction ............................................................ 43
  3.2 Background .............................................................. 43
  3.3 Factor Graphs ........................................................... 45
    3.3.1 Layered Representation .......................................... 47
  3.4 Inference on Layered Graphs ......................................... 50
    3.4.1 Max-product Message Passing .................................... 50
    3.4.2 Inference in Local Junction Trees .............................. 54
3.5 Discussion

3.5.1 Enforcing Adjacency Constraints on Delaunay Triangulations

3.5.2 Relation to Factorial Markov Random Fields

3.6 Applications

3.6.1 Segmentation

3.6.2 Defogging

3.7 Conclusion

4 Factor Graphs for Pixelwise Illuminant Estimation

4.1 Introduction

4.2 Background

4.3 Review

4.4 Illuminant Estimation

4.4.1 Background

4.4.2 Factor Graphs

4.4.3 Priors and Probabilities

4.4.4 Inference Process

4.5 Implementation

4.5.1 Initialisation

4.5.2 Illuminant Recovery

4.6 Experiments

4.7 Conclusion

5 Factor Graphs for Evidence Combining in Computer Vision

5.1 Introduction

5.2 Background

5.3 Review

5.4 Factor Graphs

5.4.1 Simplicial Complexes

5.4.2 Simplicial Spanning Tree

5.4.3 Layered Factor Graphs

5.4.4 Message Passing

5.4.5 Prototype Matching Across Layers

5.5 Implementation
<table>
<thead>
<tr>
<th>5.6</th>
<th>Applications</th>
<th>117</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.6.1</td>
<td>Defogging</td>
<td>118</td>
</tr>
<tr>
<td>5.6.2</td>
<td>Image Segmentation</td>
<td>123</td>
</tr>
<tr>
<td>5.7</td>
<td>Conclusion</td>
<td>128</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6</th>
<th>Conclusion</th>
<th>129</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Contributions</td>
<td>129</td>
</tr>
<tr>
<td>6.2</td>
<td>Future Work</td>
<td>131</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>Appendix A</th>
<th>133</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1</td>
<td>List of publications</td>
<td>133</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th>Appendix B</th>
<th>135</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.1</td>
<td>Message passing for Figure 2.4(a)</td>
<td>135</td>
</tr>
<tr>
<td>B.2</td>
<td>Message passing for Figure 2.4(b)</td>
<td>138</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>Appendix C</th>
<th>141</th>
</tr>
</thead>
<tbody>
<tr>
<td>C.1</td>
<td>Reparameterisation of a junction tree</td>
<td>141</td>
</tr>
</tbody>
</table>
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Graphical model examples. Left: Directed graph or Bayesian network. Middle panel: Undirected graph or Markov random field. Right: Factor graph.</td>
</tr>
<tr>
<td>2.2</td>
<td>Left: Tree-structured factor graph where interrelated variables (round nodes) are connected to factors (square nodes). Variable-to-factor message from Eq. (2.19). Right: The message sent from factor nodes to variable nodes in the graph according to Eq. (2.18).</td>
</tr>
<tr>
<td>2.3</td>
<td>An example factor graph encountered in computer vision. The grey nodes represent the known image tokens and the blue nodes the “hidden” variables. The black nodes are the joint probability factors. Middle and right-hand panel: Message passing from factors to variables and vice-versa.</td>
</tr>
<tr>
<td>2.4</td>
<td>Left panel: Tree-structured factor graph from Figure 2.2 with example message passing. Right panel: The factor graph as a clique tree.</td>
</tr>
<tr>
<td>2.5</td>
<td>Left: Factor graph of cycles from Figure 2.3 as a triangulated graph with edges from variable elimination. Right: The factor graph as a clique tree.</td>
</tr>
<tr>
<td>2.6</td>
<td>Common graphical representations of systems with multi-level interactions. Left: The line process MRF [124] that introduced coupled MRFs. Middle: Factorial MRF for fog removal [135]. Right: A multilayer graphical model for tracking [2].</td>
</tr>
<tr>
<td>2.7</td>
<td>Factor graph examples from the main applications listed in Section 2.5. Left: Depiction of a factor graph from decoding. Middle: Forney-style factor graph encountered in wireless networks [186]. Right: Factor graph from biological systems [176].</td>
</tr>
</tbody>
</table>
3.1 A factor graph defined over an image. Here, the variables are given by the pixels in the image lattice, which induce a cyclic arrangement of variables (round nodes) and factors (squared nodes).

3.2 Two sets of observables $\tau$ (pixel values, feature vectors etc.) corresponding to superpixels in an image. Note that the resulting structure is arranged in two layers $\gamma$, of irregular regions connected through factor nodes.

3.3 Left-hand panel: A factor graph realised in layers; Right-hand panel: Detail of the area outlined on the left-hand panel of the graph showing the beliefs sent to and from variables along the graph edges. Note that in this figure we have used $\gamma'$ to represent another layer different from $\gamma$ and $\phi$.

3.4 From left-to-right: Delaunay triangulation over a set of image regions, graph partition yielded by the normalized cuts algorithm and resulting junction tree computed from the sub-graph in the middle panel.

3.5 Algorithm steps. Left: Input image. Middle left: The input image after rescaling to half its original size and back to full size. Middle right: The image, this time scaled to a quarter of its size and back to full size. Right: Over-segmented image by oriented watershed transform [11], with region centroids in red.

3.6 Segmentation results. Left-to-right: Sample images from the BSDS500 dataset [125]; Segmentation yielded by the method in [49], Normalised cuts [161], loopy belief propagation [192] and our factor graph approach.

3.7 Precision-recall for the segmentation results yielded by our factor graph method and the algorithms in [49, 161, 192, 40]. The plots have been obtained following [11] using the human-labelled ground-truth of the BSDS500 dataset.
3.8 **Defogging results.** Left-hand column: Real-world images used in our experiments. Middle-left column: Results delivered by the method in [135]. Middle-right column: Albedo and depth obtained using the method in [167]. Right-hand column: Albedo and depth yielded by our factor graph approach. 69

4.1 Left-hand panel: Example real-world image with multiple illuminants; Right-hand panel: Colour corrected result obtained by removing the effects of the illuminants. 74

4.2 The effect of scale on the illuminant colour. Top row: Input image at a fine (left-hand panel) and coarse (right-hand panel) scales, respectively; Bottom row: Illuminant colour map yielded by the method in [68] for the images in the top row. 77

4.3 Left-hand panel: Diagram of the factor graph used throughout the chapter showing the relationships between subgraphs and pixels at different scales; Right-hand panel: Detail of one of the subgraphs in the left-hand panel, where the variables are shown as round markers whereas the factors are depicted as squared tokens. 79

4.4 Left-hand panel: Factor graph where interrelated variables (round nodes) are connected to factors (square nodes); Right-hand panel: Incoming and outgoing messages along edges between variable and factor nodes within the same scale and nodes in adjacent scales in the graph. 84

4.5 Message passing operation between nodes and factors in our graph. 86

4.6 Top row: Example images from the Gijsenij *et al.* [68] (left-hand panel) and the Beigpour *et al.* [19] dataset (right-hand panel); Bottom row: Example image from the Bleier *et al.* [26] (left-hand panel) and our dataset (right-hand panel). 93

4.7 Kernel comparison results. Performance (mean angular error in degrees) as a function of the datasets under study for our factor graph method when a number of different kernels are used. 95
4.8 Top row: Real-world input image; Middle row: Illuminant map
recovered by applying our method to the images on the top row;
Bottom row: Colour corrected images computed using the colour
maps on the middle row. ................................. 96

5.1 Example of a factor graph which can be converted into a junction
tree. The left-hand panel depicts a grid-like factor graph showing
the relationship between tokens. On the middle panel, we show the
corresponding chordal graph yielded by the approach developed
here. The right-hand panel shows the junction tree arising from the
cliques obtained by variable elimination and joined by a maximum-
spanning-tree algorithm. ................................. 103

5.2 Example representation of a factor graph as a simplicial spanning
tree. The left-hand panel shows the factor graph introduced above.
The central panel shows the triangulation obtained by computing
a convex hull on the variables spatial positions. On the right-
hand panel, we show the simplicial spanning tree arising from the
cliques obtained via a Delaunay triangulation of the graph grouped
as a maximum-spanning-tree. In the figure, the update messages
passed along edges between nodes in the graph are also shown. . . 105

5.3 Example of the interactions across layers in a factor graph with
variables shown as round markers and factors as squares. ......... 108

5.4 Simplicial spanning tree where clique variables (elliptical nodes)
are connected to neighbouring cliques through separator sets (in
parenthesis). The evidence-combined estimate from the geometric
mean is denoted in squared braces. .......................... 111

5.5 Deffoging results. Top row: Example foggy images; Second
row: Depth and albedo yielded by the method in [135]; Third row:
Results delivered by the method in [167]; Bottom row: Depth and
albedo results yielded by our evidence combining factor graph. . . 122
5.6 **Segmentation results.** Left-hand panel: Image from the BSDS500 dataset [125]; Central panels, from top-to-bottom: Segmentation results delivered by the methods in [49] and [40], respectively; Right-hand panel: Result yielded by our evidence combining factor graph. 

5.7 **Comparative results.** Left: Sample images from the BSDS500 dataset [125]. Middle: Low treewidth subtrees segmentation result from Chapter 3. Right: Segmentation result from our evidence combining approach.

5.8 Performance of the evidence combining factor graph presented in this chapter compared with the low treewidth subtrees in Chapter 3 and the segmentation algorithms in [40], [192] and [49]. The plots are also measured against human ground-truth, following the methodology in [147].
LIST OF FIGURES
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Mean and median per-pixel Euclidean angle error (in degrees) for our method and the alternatives when applied to the natural images in Gijsenij et al. [68] dataset. The median is shown in parenthesis and the absolute best performance is written in bold font.</td>
<td>90</td>
</tr>
<tr>
<td>4.2</td>
<td>Mean and median per-pixel Euclidean angle error (in degrees) for our method and the alternatives when applied to the natural images in the Beigpour et al. [19] dataset. The median is shown in parenthesis and the absolute best performance is written in bold font.</td>
<td>91</td>
</tr>
<tr>
<td>4.3</td>
<td>Mean and median per-pixel Euclidean angle error (in degrees) for our method and the alternatives when applied to the Bleier et al. [26] dataset. The median is shown in parenthesis and the absolute best performance is written in bold font.</td>
<td>91</td>
</tr>
<tr>
<td>4.4</td>
<td>Mean and median per-pixel Euclidean angle error (in degrees) for our method and the alternatives when applied to our dataset. The median is shown in parenthesis and the absolute best performance is written in bold font.</td>
<td>91</td>
</tr>
<tr>
<td>4.5</td>
<td>Mean Euclidean angle error (in degrees) when a number of different kernels are used in our method as applied to the datasets under study. Absolute best performance per dataset is shown in bold font.</td>
<td>94</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

Factor graphs are graphical models widely used for decoding noisy communication channels and signals. As graphical models, they originate from the neural networks and information theory fields as relational structures of underlying components in a system. From an algorithmic perspective, their rise is attributed to the sum-product algorithm, now known to be an instance of belief propagation in statistics and artificial intelligence [108]. Besides advancing a unified pictorial description to graphical models, it is in the use of the sum-product framework and its variations that factor graphs provide greater advantages over other graphical models. This is because the algorithm design is such that functions are evaluated locally, making computations more efficient. The method is also certain to converge when instantiated in a tree-shaped graph [90, 108].

In this thesis, we apply factor graphs and the sum/max-product algorithm to computer vision and image processing. So far, their use in this field has been limited to a handful of applications, including segmentation [159, 195], image feature fusion [97] and tracking [2]. The main reasons for this are that many of the graphs encountered in vision problems with multiple levels of interaction, such as those examined in this thesis, are computationally demanding. Moreover, the graphs are usually cyclic and the known method of converting them into trees is intractable. To this end, this thesis specifically considers a probabilistic setting to capture the physical meaning and spatial relationships of image data. We offer a version of the sum-product algorithm for multiple interactions that ensures the inference is tractable. In addition, we introduce an alternative and novel reparameterisation that transforms graphs into trees. A prominent and
recurring feature to the ideas presented here is the use of Delaunay triangulations to enforce adjacency constraints in Chapters 3 and 4, and to activate the tree reparameterisation of the graph in Chapter 5. Therefore, the main purpose of this thesis is to illustrate how the additional modelling capabilities provided by factor graphs can be successfully exploited to improve inference in computer vision problems.

1.2 Outline

The next chapter reviews the literature on factor graphs and their applications. More specifically, we discuss the factor graphs derived from probabilistic graphical models and how they have been used to obtain maximum a-posteriori (MAP) probabilities. Both exact and approximate methods are surveyed. Since, in computer vision, factor graphs with multi-level interactions have been used sparingly, we provide a brief overview of existing literature. A more in-depth review of multilayered graphs is, however, presented in Chapter 3. As such, the bulk of the material here will focus on drawing similarities between the undirected and directed graphical models that can be turned into factor graphs. We also briefly outline their connection to neural networks. In the last section of the chapter, we explore subject areas that have successfully implemented factor graphs in detail.

In Chapter 3 we introduce an approach to inference based on local subtrees of low treewidth for factor graphs with multiple layers. The layered representation is built from either one-to-one correspondencies across layers, or even many-to-one or one-to-many relationships. We impose arbitrary adjacency constraints on irregular regions within layers using Delaunay triangulations. Moreover, we partition the factor graph arising from these adjacency constraints into local subgraphs using a graph partitioning technique. Inference is then carried out by converting local sub-graphs into junction trees. We demonstrate performance using a popular segmentation database, and compare against other graph-based segmentation methods. In addition, we use the algorithm for single image enhancement in this chapter.

Chapter 4 adopts the convex hull algorithm to construct Delaunay triangulations. This is because the algorithm employed in Chapter 3 to establish this
structure frequently breaks down for grid-structured graphs, as those encountered in Chapter 4. Furthermore, in Chapter 4 we perform inference on the whole factor graph, as opposed to the local subtrees engaged in Chapter 3. To this end, Chapter 4 proposes a multiscale approach to recover the pixelwise illuminant colour for scenes lit by multiple lights. The method is based on using existing single illuminant recovery methods to obtain illuminant prototypes, along with an evidence combining step. More specifically, each single illuminant method is used to recover pixelwise illumination using multiple scales, with each scale based on a Gaussian filtered version of the original image. A multiscale factor graph is created, and MAP inference is used to estimate posterior probabilities for the subgraphs. The resulting pixel illuminant estimates from all methods are combined using a geometric mean to obtain the final result. In the chapter, we also evaluate our results and compare with existing methods. The comparison is done over three available datasets, and our new dataset of image scenes under multiple light sources. We also present recovered pixelwise illumination maps, and the colour corrected output for real-world images.

In Chapter 5, we expand and generalise evidence combining using factor graphs. Our method also uses the convex hull of Delaunay triangulations from Chapter 4 to reparameterise a grid-structured factor graph in each layer as a quotient expressing this tree-structured nature. This is similar to the parameterisation of the junction tree which computes exact inference of the graph. Computing exact inference on our tree is tractable, unlike in junction trees where it is known to be NP-hard. Furthermore, we avoid degenerate cases by adding prototype matching to align posterior outputs from other layers to the method with the most entropy. This matching is implemented using a Procrustean transformation. Lastly, we also demonstrate the utility of our method for segmentation and defogging as in Chapter 3.

Chapter 6 concludes this thesis with a summary of the developments presented here and a discussion on future research directions.
Chapter 2

Literature Review

This chapter surveys the literature that has led to the development and use of factor graphs for inference today. We begin with an introduction on the related work on inference in graphical models in general and then discuss how it has been performed in graphs commonly encountered in computer vision. In particular, Sections 2.2.1 and 2.2.2 investigate the inference in Markov random fields and factor graphs, respectively. Section 2.3 examines different multi-level graphical models that can be converted into factor graphs. We continue on and review how factor graphs have been related to traditional statistical pattern recognition systems that are modelled by neural networks. The final section investigates some key areas where factor graphs have been applied. Specifically, literature on coding, wireless networks and biological systems is reviewed in some detail. The last part of the discussion, Section 2.5.4, is devoted to outlining other application areas of factor graphs, and also concludes this chapter.

2.1 Background

This thesis focuses on carrying out probabilistic inference in factor graphs which has its origins from the inference of graphical models. Graphical models and their methods of inference were developed largely in the area of statistical physics [138, 110]. In this area, the graph represents a system of gas or solid particles such that each particle can interact with other particles nearby. For example, a model for gas particles adheres to the assignment \( x_i = 1 \) when a particle resides at a site or region \( i \). Therefore, a particle is analogous to a node that can assume different states. On the other hand, interactions are analogous to links or edges that join the neighbouring interrelated variable nodes. As is the norm in physics,
the system is modelled in terms of its entire energy $E(x)$, i.e.

$$P(x) = \frac{1}{Z} \exp \left( -E(x) \right),$$

where $P(x)$ is the joint probability distribution of the particles and $Z$ is the partition function that corresponds to a normalisation parameter. The probability distribution given by this function is called the Gibbs measure, otherwise known as the Boltzmann distribution in statistical mechanics [101].

Mathematically, a graphical model $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined by a node set $\mathcal{V}$ of variables $x_u$ and an edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ that associates nodes with each other, i.e. $(x_u, x_v) \in \mathcal{E}$. The inference for a graph $\mathcal{G}$ is dependant on the characterisation of its underlying probability distribution $P(x)$ and the variable node interactions it describes. For instance, the distribution may characterise nodes based on the direct effect they have on neighbouring nodes. If this is the case we have a directed graph or Bayesian network [138], so that $(x_u, x_v) \in \mathcal{E}$ and $(x_v, x_u) \notin \mathcal{E}$.

To this end, a Bayesian network encodes direct dependencies defined by conditional probability distributions in a way that restricts cycles. These dependencies imply independence assumptions between “parent” and “child” variable nodes in the graph. For example, the Bayesian network in Figure 2.1(a) depicts the joint distribution of five variables, i.e. $P(x_1, x_2, x_3, x_4, x_5)$. Using the conditional probability definition repeatedly, the joint probability of the graph can be simplified
§2.1 Background

\[ P(x_1, x_2, x_3, x_4, x_5) \]
\[ = P(x_1 | x_2, x_3, x_4, x_5) P(x_2, x_3, x_4, x_5) \]
\[ = P(x_1 | x_2, x_3, x_4, x_5) P(x_2 | x_3, x_4, x_5) P(x_3, x_4, x_5) \]
\[ = P(x_1 | x_2, x_3, x_4, x_5) P(x_2 | x_3, x_4, x_5) P(x_3 | x_4, x_5) P(x_4 | x_5) P(x_5) \]
\[ (2.2) \]

While the decomposition in the final expression of Equation (2.2) is mathematically accurate, the graph in Figure 2.1(a) elaborates on the conditional independence assumptions that constrain the system better than this expression. That is, the following conditional probabilities illustrate the conditional independence assumptions made, so as to write

\[ P(x_1 | x_2, x_3, x_4, x_5) = P(x_1 | x_2) \]
\[ P(x_2 | x_3, x_4, x_5) = P(x_2 | x_3, x_4) \]
\[ P(x_3 | x_4, x_5) = P(x_3) \]

Naturally, the second class of graphical models is made up of graphs in which distributions describe correlations. The graph illustrations are therefore undirected, that is, both \((x_u, x_v)\) and \((x_v, x_u)\) are elements of the edge set \(E\). Undirected graphs are referred to as Markov networks or Markov random fields (MRFs) \[101, 64\] in literature. Like Bayesian networks, MRFs are defined by conditional probability distributions. Their nature, though, is such that they satisfy the Markov property on undirected graphs,

\[ P(x_u | \mathbf{x}_{\setminus x_u}) = P(x_u | \mathcal{N}_{x_u}), \]
\[ (2.3) \]

where \(\mathbf{x}_{\setminus x_u}\) denotes all variables \(V\) in the graph apart from the variable \(x_u\), and \(\mathcal{N}_{x_u}\) is the set of variables adjacent to the variable \(x_u\). This property merely states that each variable is conditionally independent of all other variables in the graph given its neighbours. It is important to note that the inference process for Bayesian networks is initiated by converting the graph into an undirected Markov
network, *i.e.* a process called moralisation \[25, 15\]. Moralisation transforms a Bayesian network into a Markov network in two steps. First, all pairs of parent nodes with the same child node, and that are not already connected, are joined together by an undirected edge. Then, all remaining directed edges are made directionless so as to drop some conditional independencies, and to complete the procedure.

The research in this thesis is centred on a universal form of undirected graphical models, factor graphs \[58, 108\]. Factor graphs do not concern themselves with directionality, but instead depict the factorisation induced by enforcing the Markov property on the Gibbs distribution in Equation (2.1). In addition, factor graph illustrations include another set of nodes, factors \(f \in \mathcal{F}\). In this thesis, we will describe them as graphs \(\mathcal{G} = (\mathcal{N}, \mathcal{E})\), where \(\mathcal{N}\) is the node set of variables and factors, *i.e.* \(\mathcal{N} = \mathcal{V} \cup \mathcal{F}\) and the edges are between variables and factors, *i.e.* \(\mathcal{E} \subseteq \mathcal{V} \times \mathcal{F}\). Later, we will provide a factorisation of the Gibbs distribution (Equation (2.4)) so that the factors contain local conditionals obtained from this factorisation. This construction is such that each local conditional distribution contains the variables connected to the corresponding factor in its function argument.

Figure 2.1 depicts examples of the directed and undirected graphical models that are integral to the discussion in this section. On the left-hand panel, Figure 2.1(a) we show directed graph or Bayesian network, where the arrows show the conditional dependencies between variables. The middle panel shows an MRF, and in the far-right panel, Figure 2.1(c) shows a factor graph. The factor graph provides a more explicit factorisation of the function relationships between variables, *i.e.* where neighbouring variables are connected through factors that symbolically incorporate the local functions.

### 2.2 Inference in Graphical Models

Inference in graphical models is concerned with deciding the state of unknown or hidden variables given a set of observations. The state of hidden variables can be determined either exactly or through approximate methods, for which the inference is aptly termed exact or approximate, respectively. To start our discussion
on these topics, note that the joint probability given by Equation (2.1) can be expressed in terms of conditional probabilities that encode the factorisation of the graph $G$. To this end, consider the Markov property for a variable $x_u \in V$ defined on its set of neighbours $\mathcal{N}_{x_u}$ in the graph $G$, i.e. Equation (2.3). This Markov property imposes the Hammersley-Clifford theorem\(^1\) on the graph $G$ and establishes the factorisation,

$$P(x) = \frac{1}{Z} \exp(-E(x)) = \frac{1}{Z} \prod_{c \in C} \exp(-E(x_c)). \tag{2.4}$$

Equation (2.4) factors the Gibbs distribution into connected subgraphs or (maximal) cliques $c \in C$, such that $x_c \subseteq x$ is the subset of variables from the subgraph. Away from statistical physics and mechanics, the exponential in Equation (2.4) is usually replaced with a single variable function, i.e.

$$\exp(-E(x_c)) \equiv \psi(x_c).$$

In computer vision and image processing $\psi(x_c)$ is referred to as a potential function. Further, the structure of the graphs encountered in this field encourages advanced factorisation in terms of a binary $\phi(x_u)$, and pairwise potential $\varphi(x_u, x_v)$, respectively, to write

$$P(x) = \frac{1}{Z} \prod_u \phi(x_u) \prod_{v \in \mathcal{N}_u} \varphi(x_u, x_v), \tag{2.5}$$

where $\mathcal{N}_u$ is the set of variables in the neighbourhood of $x_u$.

The MRFs described by Equation (2.5) originated from extending the Ising model\(^{[101]}\) of ferromagnetism. It models the electron spin of an atom as a binary random variable and constructs an MRF by defining pairwise potentials between adjacent nodes. The binary random variable represents the direction in which magnetic atoms are aligned, i.e. whether their spin is in the upward or downward directions. In ferromagnets, magnetic atoms tend to align in the same direction.

\(^{1}\)A distribution $P(x)$ over a discrete random vector $x$ is a Gibbs distribution (i.e. it is defined by equation (2.1)), if and only if it can be represented as a product of factors, one per maximal clique $c \in C$. 
If one generalises the Ising model to systems in which variables can assume more than two states, and combine its pairwise potential with a likelihood term, models described by Equation (2.5) arise. As such, a large portion of the problems in computer vision solved with the aid of probabilistic graphical model techniques do so using inference engines whose objective is consistent with Equation (2.5).

2.2.1 Inference in MRFs

As mentioned earlier, inference in graphical models can also be divided into exact and approximate methods. In the following sections, we discuss the two approaches with a particular emphasis on MRF models.

2.2.1.1 Exact inference

Exact inference is achieved efficiently through belief propagation on tree-structured graphs [138], and a combination of variable elimination and the junction tree algorithm on arbitrary graphs [158, 90]. Belief propagation acts by making use of the variable elimination in a tree. The algorithm utilises the tree-structure and eliminates variables using a repeated sum and product operation that is analogous to distributing information between variable nodes of the MRF. By scheduling propagation in this way, belief propagation in tree-structured graphs is also referred to as the sum-product algorithm [138, 108]. The standard sum-product update rule for belief propagation in a tree-structured MRF which has the factorisation in Equation (2.5) is given by,

$$
\mu_{u \rightarrow v}^{t+1}(x_v) = k \sum_{x_u} \varphi(x_u, x_v) \phi(x_u) \prod_{w \in N_u} \mu_{w \rightarrow u}^t(x_u),
$$

(2.6)

where $k > 0$ is a normalisation constant. The tree topology is a natural organisation to compute marginals $P(x_u)$, or beliefs $b(x_u)$, for each of the variables according to the expression,

$$
b(x_u) = \sum_{x^{t+1 \setminus x^{t+1}_u} = x_u} P(x^{t+1}_1, x^{t+1}_2, ..., x^{t+1}_n),
$$
where \( P(x_{t+1}^1, x_{t+1}^2, ..., x_{t+1}^n) \) is the joint distribution of the graph after the message passing of Equation (2.6) is completed. The sum-product message passing for a factor graph involves the propagation Equations (2.18) and (2.19) between factors and variables, and vice-versa, as discussed in Chapter 1. Further, we have provided an example of the operations for a factor graph in Section 2.2.2.2.

The junction tree algorithm extends exact inference computations of belief propagation to arbitrary graphs. A tree-based inference method established on a parameterisation identical to that of the junction tree is outlined in Chapter 5 of this thesis. The original junction tree algorithm includes a variable elimination step that adds edges to any cycle that has four or more variables, as shown in Lauritzen et al. [111]. The variable elimination, which can be viewed as graph transformation, produces a chordal graph from which maximal cliques and subsequently the junction tree are created. The algorithm then implements message propagation in much the same way as the belief propagation above, although in this instance, a root node needs to be specified. We have chosen not to delve into algorithmic details here as they are provided in Section 2.2.2.4. Instead, we note that constructing the chordal graph for MRFs with cycles is known to be an NP-hard problem [94, 162, 1]. Furthermore, an increase in the number of clique variables created during the variable elimination step exponentially increases computational cost. Consequently, a junction tree’s complexity can be determined by considering its treewidth\(^2\) [27]. If the graph architecture is a chain or tree, the treewidth is usually of reasonable size and the junction tree algorithm is able to compute exact marginals effectively. Such an architecture includes the non-cyclic causal network for genetic breeding [8], in which a genealogical family structure helps determine the risk that a couple will pass-on a life-threatening disease to their offspring. In addition, the junction tree algorithm can be considered to be the exact version of the generalised distributive law [6].

A more elaborate measure of complexity, based on the sum-product operations required for the junction tree, is provided in [6]. The complexity is shown to be order \( O(\sum_{u \in V} d_u^2 q_u) \), where \( d_u \) is the degree of vertex \( u \) and \( q_u \) is the number of possible states of the vertex. As the MRFs frequently encountered in computer vision contain cycles or loops, it has been necessary to consider approximate

\(^2\)The treewidth of a graph is the number of variables in its largest clique.
inference algorithms in order to avoid these challenges.

### 2.2.1.2 Approximate inference

The fundamental problem of approximate inference in graphical models is to compute approximations that resemble the true marginals or posterior probabilities. Ideally, the method for computation should converge to an approximation of reasonable accuracy, and do so at the expense of minimal computational resources and time. Research into such approximate inference methods for MRFs has led to their broad classification as either sampling or message passing methods. In this section, we survey the literature of these approximate inference methods for MRFs and give descriptions so as to develop a general understanding of their characteristics.

**Sampling methods:** Sampling methods, in particular Markov chain Monte Carlo (MCMC) [126], are one of the oldest approximate methods. A much richer treatment of the MCMC was introduced in [76], from which the Gibbs sampling variant was adopted in Geman et al. [64] for image restoration. This MCMC method acts by iteratively drawing samples $x^t$ from probability distributions $P(x_u|x_{\setminus x_u})$ of high dimensionality to replace the variable $x_u$ and form a Markov chain. In generating the Markov chain, each new sample $x^{t+1}$ is based on the current sample for all other variables. For example, if $x^0$ denotes the state of the variables from some initial distribution, then the successive samples will be as follows,

$$
\begin{align*}
    x_1^1 &\sim P(x_1^0|x_{\setminus x_1}^0), \\
    x_2^1 &\sim P(x_2^0|x_1^1, x_{\setminus x_1-x_2}^0), \\
    x_3^1 &\sim P(x_3^0|x_1^1, x_2^1, x_{\setminus x_1-x_2-x_3}^0), \\
    \vdots \\
    x_u^1 &\sim P(x_u^0|x_{\setminus x_u}^1, x_{\setminus x_u}^0) \\
    \vdots \\
    x_n^1 &\sim P(x_n^0|x_{\setminus x_n}^1),
\end{align*}
$$

(2.7)
where the subscript in $x^*_u$ indicates that only variables with an index smaller than $u$ from the distribution are considered and $x^*_{>u}$ indicates the converse. The Markov chain generated by this process can be viewed as a sequence of correlated states that are defined mostly by the transition probabilities $P(x_{>u}^{t+1}|x_{>u}^t)$ from sample $x_{>u}^t$ to $x_{>u}^{t+1}$. In addition, the conditional probabilities for an MRF on the right-hand side of Equation (2.7) are governed by the Markov property of Equation (2.3), so that,

\[
P(x_{>u}^{t+1}, x_{<u}^t) = P(x_{>u}^{t+1}|\mathcal{N}_{x_{>u}^t}) = P(x_{<u}^t|\mathcal{N}_{x_{<u}^t}).
\]

It is important to mention that the system dynamics do not change with time, i.e. it is homogenous. This process of sampling probabilities is repeated for all variables until convergence. MCMC routines are guaranteed to asymptotically converge to the distribution $P(x)$, but at great computational expense. What guarantees convergence is the annealing step which ensures that the MRF’s energy is reduced at each cycle run. In fact, simulated annealing is a stand-alone method designed on this principle. We will present this method next. For now, we proceed to mention that MCMC is often easier to implement, it is applicable to a broader range of models, such as those whose size or structure changes depending on the values of certain variables, e.g. in matching, or models without nice conjugate priors, and sampling can be faster than variational methods when applied to really huge models or datasets.

Note that the original Gibbs distribution in physics includes the system temperature $T > 0$, so that Equation (2.4) is written as

\[
P(x) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \exp\left(-\frac{1}{T} E_c(x_c)\right).
\]

Simulated annealing hinges on adjusting this temperature, usually in very small steps. When the temperature tends to zero the system is at its most probable state, making it easier to identify the global energy $E(x)$ minimum. For this reason, simulated annealing is a popular maximum a-posteriori probability (MAP) local search method in image processing. In detail, the method acts by slowly reducing the temperature with the aid of a proposal distribution $Q$ to
sample a new state. Using the energy derived from this distribution $E(x_Q)$, the algorithm evaluates,

$$\alpha = \exp \left( \frac{E(x) - E(x_Q)}{T} \right),$$  \hspace{1cm} (2.8)

where $x_Q \sim Q(\cdot|x^t)$ is a sample generated when the $t^{th}$ sample of $x$, i.e. $x^t$, is perturbed with a normally distributed perturbation $\epsilon^t$, i.e.

$$x_Q = x^t + \epsilon^t, \hspace{0.5cm} \text{where } \epsilon^t \sim N(0, \Sigma).$$

After this step, the method decides whether to adopt the proposed state (i.e. set $x^{t+1} = x_Q$) with probability $\min(1, \alpha)$, or continue with the current state (i.e. set $x^{t+1} = x^t$). It is obvious from the expression in Equation (2.8) that the temperature value determines whether or not a new state is adopted. A high temperature increases the likelihood to adopt a change of state and when the temperature drops, it is less likely to change states. In fact, the algorithm converges to the global optimal solution if the temperature is annealed (i.e. cooled) sufficiently slowly [102]. Simulated annealing is, however, a slow and computationally demanding method since it is difficult to define annealing steps that lead to convergence [130]. Furthermore, convergence is also heavily dependant on the nature of the proposal distribution. This is why message passing algorithms have become the preferred methods of inference for MRFs in recent times.

**Message passing methods:** Message passing algorithms are a class of inference methods that often converge faster than sampling methods and deterministic annealing. Unlike other methods, they provide easier means to determine convergence. They can be further categorised into either variational or local search classes. The tutorials [95, 181] present excellent formalism of variational methods. So far, we have already mentioned belief propagation for tree-structured graphs as a sum-product rule in Section 2.2, so that here we can consider this variational method for MRFs with cycles. When belief propagation is used for approximate inference in cyclic MRFs, it is known as loopy belief propagation (LBP) [138, 50]. The cycles or loops in a graph change the message passing scheme as there are neither leaf nor root nodes to initiate message propagation. Instead, messages are distributed according to the scheduling in Section 2.2.2.3 until a user-defined...
termination criteria is met. This makes it difficult to determine convergence, and in some cases the method does not converge or does not produce a result [194]. By relating the optimal points of LBP to those from the Bethe free energy approximation, the conditions under which LBP will converge were shown in [194]. Yedidia et al. define the Bethe free energy $F_{\text{Bethe}}$ as the difference between the Bethe average energy $U_{\text{Bethe}}$ and the Bethe entropy $H_{\text{Bethe}}$, i.e.

$$F_{\text{Bethe}} = U_{\text{Bethe}} - H_{\text{Bethe}}.$$  

(2.9)

The Bethe average energy and entropy are defined as follows,

$$U_{\text{Bethe}} = -\sum_u b(x_u) \ln \phi(x_u) - \sum_{v \in N_u} b(x_u, x_v) \ln \varphi(x_u, x_v)$$

$$H_{\text{Bethe}} = -\sum_{u,v} \sum_{x_u, x_v} b(x_u, x_v) \ln b(x_u, x_v) + \sum_u (d_u - 1) \sum_{x_u} b(x_u) \ln b(x_u),$$

where $b(x_u)$ and $b(x_u, x_v)$ are the beliefs encountered in MRFs from Equation (2.5). The other variables are as before; i.e. $\phi(x_u), \varphi(x_u, x_v)$ being potentials and $d_u$ is the degree of vertex $u$. Further, by adding Lagrange multipliers to $F_{\text{Bethe}}$ and taking derivatives wrt the beliefs $b(x_u), b(x_u, x_v)$, they show that the stationary points of the Lagrangian equate to the belief propagation equations. As such, belief propagation converges to the fixed points of the Bethe free energy approximation. More importantly, this means that the convergence conditions of the Bethe free energy correspond to conditions under which belief propagation converges. Around the same time, sufficient conditions for convergence were derived in [170, 127]. Tatikonda [170] related convergence rates to the uniqueness of the Gibbs measure determined by Dobrushin’s condition³ [65], and concluded that rates are also exponential in the number of times the cycles are repeated during propagation. The conditions were improved upon in [127], where a general convergence criteria is given in terms of the spectral radius of the matrix induced by the contraction mapping of the messages. Moreover, convergence is shown to be independent of the initial messages. Damping, and several variants to message scheduling that improve convergence have also emerged [129, 180, 48]. In prac-

³Dobrushin’s condition is a classical condition that defines the contraction criteria to ensure uniqueness of Gibbs measures.
tice, the sum in LBP can be substituted by a maximum to yield a special case of the algorithm called the max-product algorithm \[138\], which has the update equation

\[
\mu_{u \rightarrow v}^{t+1}(x_u) = \max_{x_u} \varphi(x_u, x_v) \phi(x_u) \prod_{w \in N_u} \mu_{w \rightarrow u}^{t}(x_u),
\]

(2.10)

when the summation in Equation (2.6) has been substituted by a maximum, and we have omitted the normalisation constant. This max-product is the predominant method in computer vision to compute MAP estimates, and subsequently the most probable state of variables \(x^*\).

Another variational method is the mean field \[139, 137\] which also originated from statistical physics as a tool for Boltzmann machines. The mean field method factorises an approximate distribution \(Q\) using a collection of distributions that are similar, and on which inference is easily computed, i.e.

\[
Q(x) = \prod_i Q_i(x_i).
\]

The goal is then to minimise the distance between the approximate distribution \(Q(x)\) and the target distribution \(P(x)\), which is appropriately measured by the Kullback-Leibler (KL) divergence defined as follows

\[
KL(Q|P) = \sum_u Q(x_u) \log \frac{Q(x_u)}{P(x_u)}.
\]

Although these evaluations can be carried out efficiently, the ease in computation is at the expense of accuracy. A way to alleviate this is to add more information to the model, from which more accurate deductions can be inferred. To this end, Ghahramani et al. \[67\] captured additional information using factorial hidden Markov models, which are the foundation of the layered structures in this thesis. The mean field approximation in these graphs was then obtained in much the same way as the Bethe free energy approximation \[193\].

**Local search methods:** Local search algorithms are designed for MAP inference that attributes each graph node a most likely state after optimising the energy function in Equation (2.1). Iterated conditional modes (ICM) \[22\] is a
classical local search method which assigns a state to a node given its neighbours;

\[ x_u^* = \arg\max_{x_u \in \mathcal{V}} P(x_u | \mathcal{N}_u). \]

Each node is assigned an initial state before inference begins for all nodes. Subsequent estimates are dependent on previous states, and the algorithm converges when consecutive iterations do not result in a change of state. This dependence between iterations, however, makes the method susceptible to terminating under local minima conditions.

Recently, graph cuts \[31, 105\] have emerged as an important and popular local search method in the vision community. Graph cuts use maximum flow algorithms for graphs to minimise the energy function in Equation (2.1). The method involves constructing a directed graph with distinct vertices, a source node \(s\) and a sink node \(t\). The edges in the graph carry a weight, and the objective is to partition the graph into disjoint sets \(S\) and \(T\) at minimal cost. The cost here is the sum of all edge weights that are set apart because of the cut, \(i.e.\) for nodes \(u \in S\) and \(v \in T\) after the cut, the cost \(c(S, T)\) of dividing the graph into \(S\) and \(T\) is

\[ c(S, T) = \sum_{(u, v) \in \mathcal{E}} w(u, v), \]

where \(w(u, v)\) is the weight of the edge connecting nodes \(u\) and \(v\). Moreover, the cut should ensure \(s \in S\) and \(t \in T\). Finding the \(s/t\)-cut with the least cost is equivalent to evaluating the maximum flow from the source node \(s\) to the sink node \(t\). Graph cuts have been shown to produce the exact global minimum for binary MRFs whose energy satisfies the so-called submodularity metric \[72, 105\], \(i.e.\) for any pair of labels \(\ell_1\) and \(\ell_2\),

\[ E(\ell_1, \ell_1) + E(\ell_2, \ell_2) \leq E(\ell_1, \ell_2) + E(\ell_2, \ell_1). \quad (2.11) \]

If the MRF-MAP problem has multiple states or labels and is submodular, exact inference has also been shown to be achievable through converting it into a binary problem \[155, 143\]. Multi-label MRF-MAP problems for general energy functions are, however, intractable for exact inference. Nevertheless, it is possible to find quality estimates with relative efficiency by taking greedy hill-
climbing steps as in [31]. In addition, Kolmogorov and Rother [104] introduced the quadratic pseudo-boolean optimisation (QPBO) method, a linear programming (LP) relaxation technique, to extend graph cuts to non-submodular functions. The QPBO algorithm [104, 153] works by solving an LP relaxed version of the energy function which, first, reparameterises the energy terms into a normal form. Then, evaluating the maximum flow to obtain the minimum $s/t$-cut on the graph gives a partial labelling which is part of the global optimal solution. Several options on how to deal with the unlabelled nodes exist; including either for the nodes to retain their old labels, or to run the max-product belief propagation on them. Another relaxation is the truncation method [154] that interchanges non-submodular parts of the function with a submodular approximation. That is, if an energy term $E_{u,v}$ between nodes $u$ and $v$ does not satisfy the submodularity condition in Equation (2.11), then for any pair of labels $\ell_1$ and $\ell_2$ either: one of the terms $E(\ell_1, \ell_2)$ or $E(\ell_2, \ell_1)$ is increased, or the term $E(\ell_1, \ell_1)$ is decreased until the condition is satisfied.

A detailed and advanced review of many of the methods presented above can be found in [166, 98]. In the next section, we will devote our attention to inference in factor graphs, which can be used to depict any directed or undirected graphical model.

### 2.2.2 Inference in Factor Graphs

Any graphical model can be converted into a factor graph, which means the inference algorithms presented above are equally applicable to these graphs. The work on factor graphs by Frey et al. [58, 57] expresses the conversion of several common graphical models into factor graphs well. Indeed some of the inference routine citations in Sections 2.2.1.1 and 2.2.1.2 pertain to literature on factor graphs. This research is concerned with probabilistic inference as a problem to attribute variables to particular labels or classes in a MAP setting.

In the following sections we put forward preludes to factor graphs, the operations of the sum-product algorithm as applied to both tree-structured graphs and arbitrary graphs, and the junction tree algorithm.
2.2.2.1 Background

Factor graphs are useful in modelling multivariate functions of variables that can be expressed in a factorised product form. They consist of two sets of nodes, variables and factors. In computer vision, probability mass functions are used to describe the inter-connected states of variables that represent the workings of the system. In such cases factor graphs capture the relationships between variables through conditional probability distributions defined in factor nodes. More formally, a factor graph is defined as a graph $G = (N, E)$ comprising of a node set $N$ and an edge set $E$. Its node set can be further decomposed into distinguishable subsets of variables $V$, and another for factors $F$, such that $N = V \cup F$. The factor node moderates the influence of surrounding variables via the local conditional probability distribution associated to it. On the other hand, the edge set connects variables to factors, thereby parameterising the conditional probabilities defined by the graph. Here, we view this composite function of the graph as a factorisation of the Gibbs distribution $P(x)$ over the set of variables $x$. In a factor graph, the Gibbs distribution is considered on the factors and we can refer to the factors as cliques, and call their conditional distributions clique potentials or potential functions $\psi(\cdot)$. In this manner, we can define the graph’s composite function as follows

$$P(x) = \frac{1}{Z} \prod_{c \in C} \psi(x_c), \quad (2.12)$$

where $Z$ acts as a normalisation constant. Equation (2.12) defines the composite function in terms of potential functions at each of the nodes $x_c$ corresponding to the subset of variables comprising the clique $c$. This Gibbs measure is related to the Boltzmann distribution [101] from statistical physics.

In practice, the left-hand side of Equation (2.12) is a joint distribution over the observables $\tau$, i.e. image features, pixels, superpixels, etc., and the state of variables $x$, such that

$$P(x) \propto P(x, \tau). \quad (2.13)$$

By definition of the chain rule of probability, this can be simplified to yield

$$P(x, \tau) = P(x|\tau)P(\tau), \quad (2.14)$$
so that we can write

\[ P(x|\tau) = \frac{1}{Z} \prod_{c \in C} \psi(x_c; \tau_c), \]  

(2.15)

in terms of the clique potentials \( \psi(x_c; \tau_c) \), and \( Z = P(\tau) \) is a constant defined over the set of observables.

Continuing from Equation (2.15), notice that we can use Bayes’ rule to express the clique potentials in terms of a likelihood and a prior, i.e.

\[ P(x|\tau) = \frac{1}{Z} \prod_{c \in C} \psi(x_c; \tau_c) = \frac{1}{Z} \prod_u P(\tau(u)|x(u))P(x(u)) \]  

(2.16)

The token \( \tau(u) \) becomes a vertex location in the graph \( G \) such that \( x(u) \) denotes the variable attributes at a vertex \( u \). Notice that \( Z \) corresponds to the marginalisation of the numerator in Equation (2.16). This is equivalent to the marginal distribution \( P(\tau) \) over the entire graph.

Factor graphs, and graph models in general, are constructed to identify and interpret the machinery involved in a system and how the system works. In a probabilistic setting, this equates to inferring the probabilities of interest and deducing the characteristics of the underlying system. One such inference problem utilises Bayesian statistics to estimate the maximum a-posteriori probability (MAP) or most probable state of a system. Using the derivations from the previous section, the problem can be stated as,

\[ x^* = \arg\max_x P(x|\tau) = \arg\max_x \prod_u P(\tau(u)|x(u))P(x(u)), \]  

(2.17)

where \( x^* \) denotes the most probable state of the variables, and the final expression has made use of Bayes’ rule from Equation (2.16). A graph provides a visualisation of the problem so that engineers understand better the problem simplifications and how best an inference algorithm can be developed. Existing inference algorithms for graphical models were motivated by trends in coding theory.
Inference algorithms that are used for factor graphs today were derived from the Viterbi algorithm [55] developed for finite-state discrete-time Markov processes. In that work, a trellis structure of a digital communications problem is cast into a MAP framework to determine its most likely state sequence. In a similar vein, Bahl et al. [14] presented a forward-backward decoding algorithm for a maximum likelihood Markov system. A trellis also diagrammatically represented the time evolution of the state sequences encountered in the problem. Further improvements to these algorithms led to the development of the min-sum and sum-product algorithms for iterative decoding [184, 185]. Their graphical illustrations translated parity-check matrices into generalised trellis structures, i.e. Tanner graphs [169]. The existing sum-product algorithm for factor graphs [58, 108] exploits the bipartite graph construction of Tanner graphs to employ a forward-backward recursive inference procedure. The sum-product algorithm for factor graphs establishes a unifying standard to describe the message propagation of all these algorithms.

2.2.2.2 The sum-product algorithm

We now turn our attention to the Gibbs distributions as factorised over the clique potentials given in Equation (2.16). This factorisation allows the construction of a factor graph structure with factors composed using the probability distributions in Equation (2.16). Probabilistic inference using the sum-product rule entails recovering the marginal probabilities \( P(x(u)) \) of all the variables at the factors. This is accomplished by utilising an organised way to exchange probability information between variables as “messages”. To this end, the sum-product algorithm is often referred to as a message passing scheme. Note that from the marginals computed by the sum-product algorithm, we can obtain the MAP assignment for all variables \( x^* \) by taking the state with the highest probability. MAP formulations such as the one given in Equation (2.17) are the fundamental problems solved in this thesis. For now we will use Figure 2.2, which shows an example factor graph, to illustrate the message passing procedure which computes marginal probabilities of a small set of variables.

Figure 2.2 shows a set of variables arranged in a tree structure. In the figure, and for the sake of clarity, we have adopted the following notation. For the
variables we have used \( x \) and \( y \). We have denoted the factors as \( f \) and \( g \), and used \( \mu \) to indicate a message flowing in the direction of the sub-indexed arrow.

Note that, from the figure, we can observe that the factor node \( f \) “passes on” to a variable node \( x \), the message

\[
\mu_{f \rightarrow x} = \sum_{x' \in X \setminus x} F(x') \prod_{y \in N_f, g \neq f} \mu_{g \rightarrow x},
\]

where \( X = N_f \) denotes the set of all variable nodes adjacent to the factor \( f \) and \( F(x') \) is the conditional probability at the factor.

Moreover, the variable-to-factor interactions can be expressed explicitly as follows

\[
\mu_{x \rightarrow f} = \prod_{g \in N_x, g \neq f} \mu_{g \rightarrow x},
\]

where \( N_x \) is the set of factor nodes neighbouring the variable node \( x \).

In order to compute the marginal probabilities \( P(x|u) \), messages are first sent from the leaf nodes to neighbouring nodes simultaneously. If the leaf node carries no information on the nature of the probabilities that constitute the message, then an “identity function” of the probability is used, for example, a uniform distribution. Once all messages have propagated along edges connecting the vertices neighbouring leaf nodes, a second set of messages can now be computed and passed to the edges that are in the graph’s interior. The same procedure
§2.2  Inference in Graphical Models

follows in successive message computations; that is, before a node sends a message along a given edge, it waits to receive messages from all other neighbouring nodes. This process is repeated until each edge has a message flow through it in either direction, upon which the algorithm terminates. Finally, the marginal distribution or belief \( b(x(u)) \) of a variable \( x(u) \) is obtained by the product of all messages flowing into it, \( i.e. \),

\[
P(x(u)) = b(x(u)) = \prod_{f \in \mathcal{N}_x} \mu_{f \rightarrow x}.
\]  

(2.20)

This algorithm was described in [108] as the single-\( i \) algorithm to compute a marginal function for a single variable \( i \).

### 2.2.2.3 Loopy belief propagation

However, the factor graphs often encountered in computer vision are graphs with cycles. Figure 2.3 depicts a typical example of the type of grid-structured factor graph encountered in computer vision or image processing problems. In the figure, we denote an unknown variable as \( x_u \) to indicate that it is associated with the known image token \( \tau(u) \). We have adopted this notation here as we will discuss a message passing between variables and sub-indexing with \( u \) is a common adaptation. We also write, in short, \( x(u) \equiv x_u \). Here, the same message propa-
gation equations (2.18), (2.19) from the tree-structured graph in Figure 2.2 apply. Figure 2.3(b) shows the message passing of equation (2.18) and Figure 2.3(c) is a pictorial description of Equation (2.19). In the first instance, information is sent from the factor nodes to the variable nodes. For example, the “likelihood” of the unknown variable \( x_u \) given the observed token \( \tau(u) \), is sent to the unknown variables. This is equivalent to the left-hand side of Equation (2.18), with the function in the right-hand side \( F(\mathcal{X}) = P(\tau(u)|x_u) \). In the same vein, the prior factors send messages to variables when the function is defined as \( F(\mathcal{X}) = P(x_u, x_v) \).

It is important to mention that the variable-to-factor message in the equation is again initialised with an identity function or uniform distribution. The second phase of message passing involves sending messages from unknown variable nodes to the factors, which is Equation (2.19). For graphs with cycles, these two steps are repeated until a user-defined convergence criteria is met. If we combine the evolution of Equations (2.18) and (2.19) such that we have a single message between hidden variable nodes, then at iteration \( t \) we will have,

\[
\mu_{x_u \rightarrow x_v}^{t+1} = \sum_{x_u} F(x_u, x_v) \prod_{w \in \mathcal{X}_u \setminus w \neq v} \mu_{x_w \rightarrow x_u}^t.
\]  

(2.21)

The function \( F(x_u, x_v) \) is determined by both the likelihood and the prior probability, \( i.e. \)

\[
F(x_u, x_v) = P(\tau(u)|x_u)P(x_u, x_v).
\]

Summarising message passing for graphs with cycles with Equation (2.21) is known in literature as loopy belief propagation \cite{138}. The general probability propagation algorithm presented here is sometimes referred to as sum-product belief propagation.

**Improving loopy belief propagation:** It is easy to see that loopy belief propagation is different from the sum-product algorithm in its message propagation scheduling. The sum-product algorithm has messages instantiated at the leaves, while loopy belief propagation is carried out synchronously. In addition, the algorithmic steps are repeated several times until a subjective convergence condition is satisfied. For this reason, loopy belief propagation does not give exact proba-
bility marginals for graphs with cycles. Another challenge is that in practice, the convergence rate and performance of approaching inference in graphs with this topology is generally poor. Historically, this triggered the development of strategies to combat these drawbacks. One approach was damping \cite{29}. Damping seeks to ensure convergence by introducing a Lagrange multiplier $\lambda$, and substituting the message $\mu_{x_w \rightarrow x_u}$ in Equation (2.21) with a damped message; \textit{i.e.}

$$\hat{\mu}_{x_w \rightarrow x_u}^t = \lambda \mu_{x_w \rightarrow x_u}^t + (1 - \lambda) \mu_{x_w \rightarrow x_u}^{t-1},$$

(2.22)

where $0 \leq \lambda \leq 1$. This reduces oscillation in computations and improves the rate of convergence.

Damping, however, does not necessarily provide convergence guarantees nor better convergence rates in all cases. To this end, other schedule schemes which impact convergence have been proposed. Synchronous update scheduling is the most common choice of organising how messages are sent out. This involves carrying out the message updating procedure simultaneously, whether it be sending messages from variables to factors or vice-versa. On the contrary, asynchronous updating \cite{58} involves sending messages one at a time. Analysis of convergence rates showed that the contraction for asynchronous updates was superior to synchronous scheduling \cite{58}. Elidan \textit{et al.} \cite{58} also presented a scheduling that chooses to update the message which is most different from its predecessor in norm value terms. This “message norm” is called a residual, and the method of inference driven by this update method, residual belief propagation. Lastly, tree reparameterisation \cite{180} reorders update schedules by selecting subsets of spanning trees. Inference is then performed through a series of serial updates (asynchronous scheduling) on each tree while messages in other trees remained unchanged, until all sub-trees have been updated.

Here, we do not aim to exhaust the existing scheduling schemes in literature. Instead, we will proceed with the graph structure where asynchronous scheduling is most effective, the junction tree.
2.2.2.4 The junction tree algorithm

In this section, we discuss the ingredients needed to arrange factor graphs with loops into a junction tree. A junction tree \cite{Kschischang01} is a graph $G$ with vertices of cliques $C_G$ such that if a variable $x$ is present in two cliques $c_i$ and $c_j$, then all cliques on every walk between $c_i$ and $c_j$ also contain $x$. The necessary condition needed for a graph to be a junction tree is known as the running intersection property\textsuperscript{4}. In the previous section we presented the machinery for exact inference in a tree-structured graph by multiplying factors and marginalising variables. Exact inference in junction trees essentially follows the same concept in a way that renders the message passing of the sum-product algorithm to be between clique factors. In the following we will present a distinct junction tree and consider converting a loopy graph into a junction tree to illustrate this assertion.

The junction tree for tree-structured graphs: In the left-hand side of Figure 2.4 we include detailed representations of nodes and message scheduling to our example tree-structured factor graph of Figure 2.2. We have chosen to subscript the factors in accordance with the variables associated with its probability potential, e.g. $f_{23}$ represents a factor whose potential is determined by the variables $x_2$ and $x_3$, i.e. $\psi(x_2, x_3)$. Furthermore, suppose that the Gibbs measure

\textsuperscript{4}A tree-structured graph $G$ is said to have the running intersection property if all cliques, on every walk between any two cliques $c_i$ and $c_j$, contain any variable $x$ that is present in the intersection of $c_i$ and $c_j$. 

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure2.4}
\caption{Left panel: Tree-structured factor graph from Figure 2.2 with example message passing. Right panel: The factor graph as a clique tree.}
\end{figure}
given by the graph is,

$$P(\mathbf{x}) = P(x_2|x_1)P(x_3|x_2)P(x_2|x_4, x_5)P(x_4)P(x_5), \quad (2.23)$$

according to the respective factorisation of the potentials $\psi(\cdot)$. We can, therefore, define the factor potentials as follows

$$\psi(x_1, x_2) = P(x_2|x_1), \quad \psi(x_2, x_3) = P(x_3|x_2),$$
$$\psi(x_2, x_4, x_5) = P(x_2|x_4, x_5), \quad \psi(x_4) = P(x_4), \quad \text{and} \quad \psi(x_5) = P(x_5).$$

According to Equation (2.20), the marginal probabilities given by this graph are

$$P(x) = \prod_{f \in N_x} \mu_{f \rightarrow x}. \quad (2.24)$$

Equation (2.24) shows the message propagation in a tree to be exact as it is marginalisation of the global function $P(\mathbf{x})$ over the variables not being considered. The detailed exposition of the inference process for the example here, according to the message scheduling shown by the circled numbering in the figure, can be found in Appendix B.

Another intuition is to express $P(\mathbf{x})$ in Equation (2.23) as

$$P(\mathbf{x}) = P(x_2|x_1)P(x_3|x_2)P(x_2, x_4, x_5), \quad (2.25)$$

where we have used Bayes’ rule to obtain the joint distribution $P(x_2, x_4, x_5)$. The right-hand panel of Figure 2.4 shows our factor graph in a clique tree representation according to this new factorisation. It is evident that this graph satisfies the running intersection property and is therefore a junction tree.

The inference process in a junction tree is initiated through the selection of a root node. For the junction tree in Figure 2.4 let the root node be node $c_2$ with $x_2, x_3$ as variables. We have deliberately selected this leaf node so that we distinguish the inference process here from the direct sum-product in factor graphs above. Next, we initialise the clique potentials $\Psi(x_{c_i})$ of each clique $c_i$.
using the following equation

\[ \Psi(x_i) = \prod_{x_c \subseteq x_i} \psi(x_c). \]  \hfill (2.26)

The variable \( x_i \in x \) denotes the variables associated with clique \( c_i \). The message sent back and forth between two cliques \( c_i \) and \( c_j \) in a junction tree is,

\[ \mu_{c_i \rightarrow c_j}(s_{i,j}) = \sum_{c_i \setminus \{s_{i,j}\}} \Psi(x_i) \prod_{k \in (N_i \setminus \{j\})} \mu_{c_k \rightarrow c_i}(s_{k,i}), \] \hfill (2.27)

where \( s_{i,j} = c_i \cap c_j \) is the set of variables common to the adjoined cliques. The first stage of the messaging passing process sends these messages towards the root node and the second and final stage sends messages in the opposite direction.

Similar to factor graphs, we finally compute a “belief” \( B(x_{c_i}) \) for each clique after the propagations terminate, i.e.

\[ B(x_{c_i}) = \Psi(x_{c_i}) \prod_{k \in N_i} \mu_{c_k \rightarrow c_i}(s_{k,i}). \] \hfill (2.28)

The marginals for each single variable \( x \) are obtained by marginalising the beliefs of a clique containing the variable over the remainder of the clique’s variables, i.e. marginalisation over \( x_{c_i} \setminus x \). In summary, the marginals for the junction tree in Figure 2.4(b) are given by

\[ P(x) = \sum_{x_{c_i} \setminus x} B(x_{c_i}), \quad \{x : x \in x_{c_i}\}. \] \hfill (2.29)

Notice that we can compute marginals for variable \( x_2 \) using any of the three beliefs. We can also verify that the marginals obtained by this procedure are equivalent to those from the message passing described in Section 2.2.2.2 and given by Equation (2.24). Appendix B also gives a detailed account of the derivation of the clique beliefs.

In summary, the sum-product algorithm of Equation (2.27) is motivated by the first equality of Equation (2.16) for clique potentials, while Equations (2.18) and (2.19) correspond to the sum-product based on the explicit factors from the probability factorisations in the last line of Equation (2.16). As such, the message...


The junction tree for arbitrary graphs: In this section, we investigate how to convert a grid-structured factor graph into a clique tree or junction tree. We can follow the basic idea of the procedure in Section 2.2.2.2 above of summing out variables. This operation, also known as variable elimination, is performed for each variable until all variables in the graph are exhausted. However, at each summation step, a clique consisting of the variable being eliminated and all its neighbours is created. This equates to adding an edge between every pair of variables in the set of neighbouring variables. In Figure 2.5(a) we show the fill-in edges (red dashed lines) that a variable elimination sequence - which eliminates the variable with the fewest neighbours - adds to the factor graph in Figure 2.3. The neighbours and the variable in question together form a clique. Once the set of cliques is constituted, they are then joined together by a maximum spanning tree strategy. This step involves connecting any cliques $c_i$ and $c_j$ such that the cardinality, $|c_i \cap c_j|$, of the variables in their intersection is a maximum.

Consider the four-connected graph from Figure 2.3 as an example. In the left side of Figure 2.5, we show the cyclic graph along with the edges produced by the variable elimination operation. The variables are eliminated in the following order, $x_1, x_3, x_7, x_9, x_2, x_4, x_5, x_6, x_8$, which is given by choosing the variable with the least number of neighbours in the graph. The right-hand side shows the
resulting clique tree after maximum cardinality search. Another variable elimination approach is the min-fill \[151\] which eliminates a variable that adds the least number of edges to the new graph. In addition to variations of the min-fill and min-neighbours are various other greedy variable elimination methods \[103\]. The popular ones are,

- **Min-fill**: Eliminate variable which leads to the fewest fill-in edges as a result of its elimination.

- **Weighted min-fill**: Eliminate variable with the least sum of weights from its fill-in edges. Here, the weight of an edge is considered to be the product of the weight of its two vertices.

- **Min-neighbours**: Eliminate variables with the fewest neighbours.

- **Min-weight**: Eliminate variables with the least product of the vertex weights of its neighbours.

A graph \(G\) is also said to be a junction tree if and only if it is triangulated or chordal\(^5\) [15, 103]. More expressly, any triangulated or chordal graph can be straightforwardly decomposed into a junction tree. This means that by turning a graph into a chordal graph, the graph will be a junction tree. In our example, the graph obtained after variable elimination in the left-hand side of Figure 2.3 is evidently chordal with the right-hand side of the figure showing the graph as a junction tree.

It is worthwhile to mention that computing inference using the junction tree algorithm yields exact results. However, this is computationally infeasible for cyclic graphs, as transforming them into chordal graphs is exponential in the treewidth [12]. This would, therefore, be especially hard for graphs of coupled systems or with multi-level interactions. For this reason, the loopy belief propagation mentioned earlier in Section 2.2.2.3 is usually the favoured message passing algorithm to approximate probabilities. In this thesis, we address the problem through triangulations that offer variations to the sum-product algorithm and a tree-reparameterisation related to the junction tree to obtain better approximations to the exact distribution.

\(^5\) A graph \(G\) is described as chordal if any loop with four or more connected variables has a chord.
2.3 Graphs with Multi-level Interactions

As mentioned above, factor graphs allow scientists to capture additional statistics that would otherwise be unnoticed in MRFs. In this thesis, we harness this property for complex systems that involve interdependencies, hierarchies or multiple scales, to devise algorithms that are tractable. Existing graphical representations of multi-level interactions are tabulated in literature as coupled MRFs [124], factorial MRFs [100] or multi-layer MRFs [2, 132, 13, 99, 20]. Figure 2.6 shows the pictorial descriptions of the multi-level graphs from [124, 135] and [2] respectively.

The coupled MRF or line process [64, 124], as it is sometimes called, links together a binary MRF indicating the occurrence of an edge with a continuous MRF on image intensities for piecewise restoration. Coupling the image restoration problem with the binary MRF ensured edge occurrences were not oversmoothed after restoration. In the left-hand panel of Figure 2.6(a), we show the line-process MRF as a double lattice between the continuous valued MRF which is represented by the solid lines and blue-coloured variables. In-between these variables are black dots and the dashed lines due to line process, where the edges joining image intensities can be considered to be its variables. Narasimha [132] also seek to preserve boundaries when estimating disparities, although the MRF for boundaries does not use the line process. Note also that they simultaneously estimate the two in a coupled MRF that identifies boundaries using disparity.
discontinuities, and corrects the disparity using the boundaries. Coupled MRFs can also be found in segmentation [87, 188], multiple target tracking [189] and optical flow [79].

Factorial MRFs (FMRF) were introduced in [100] to mirror the progress of hidden Markov models to factorial hidden Markov models [67]. In the FMRF construction layers were assumed to be apriori independent, and their cooperation was only determined through the observed data. Inference was carried out with the help of an alternating expectation-maximisation (EM) scheme, and their usage was demonstrated using a stereo matching example. Later, Nishino et al. [135] used an FMRF to factorise a foggy image into albedo and depth layers that were jointly determined using Bayesian statistics. Figure 2.6(b) is a depiction of how the FMRF relates the albedo \( \gamma \), and depth \( \phi \), layers through the single foggy observation \( u \). Notice that the visualisation contains no links between layers since they are considered to be statistically independent. As is the case in layered representations, the method is known to be computationally demanding. A reformulation of the FMRF energy function into a sparse linear system, through a relaxation step, was proposed in [131] to reduce complexity. Schwartz and Nishino [157] chose to implement a non-linear optimiser to the original FMRF energy function instead. Both methods derive closed-form expressions for the layers that are evaluated interchangeably.

Multiple interactions reported as multi-layer MRFs have been used in motion analysis [182, 13], segmentation [99] and change detection [20]. In motion analysis, each layer is an image taken from the video sequence and is described by intensity, opacity and motion maps. In [13], the description is defined by an additive Gaussian mixture. The algorithm in [182] describes an optical flow field, determines a set of affine motions, and then computes motion parameters using linear regression. In addition, the motion parameters are iteratively refined, so that a pixel is classified into a layer based on the discrepancies between the optical flow at its location and the vector consisting of computed affine parameters. In [2], graphical model layers represent relocatable objects which cause occlusions in the tracking of a person’s motion. We have provided a graphical depiction of the dynamic Bayesian network in which relocatable and occluding objects are represented by layers in Figure 2.6(c). The graph represents a com-
plex relationship, using directed edges, in which the person $x$ being tracked can not only change location around a movable object $\gamma$ or $\phi$, but can also move through these objects over-time $t$. These occurrences are captured by the conditionals $P(x_{\gamma,t+1}|x_{\gamma,t},x_{\phi,t})$ and $P(x_{\phi,t+1}|x_{\gamma,t},x_{\phi,t})$. This is in contrast to FMRFs which have apriori independent layers. The segmentation in [99], however, uses the FMRF construction where layers interact at one combined level. There, they segment single images whose different texture features are assigned to different layers. Recently, Benedek et al. [20] have compared different multilayer MRFs for change detection in optical remote sensing images. They revealed that minimal differences existed between the various layered models if the right features and parameters were selected.

While the benefits of factor graphs are widely accepted in the community, to our knowledge, only a few have exploited factor graphs for multi-level dependencies in computer vision. Shi et al. [160] introduced them as factor metanetworks, an extension to Bayesian metanetworks\(^6\) [171]. In addition, they demonstrated their utility in data fusion of wireless sensor networks. The network consists of sensors that track objects, and different levels could have different topologies that are linked and influence each other. In their example, the objects a sensor can track in a wireless network are assumed to be affected by weather conditions. The different weather conditions serve as the layers that can be used to solve a sensor fusion problem. Following similar assumptions, Kampa et al. [97] change a dynamic-tree Bayesian network into a factor graph to carry out segmentation and data fusion at the same time. Nodes in their tree are arranged in a cascade of separate scales. Tree-structured graphs with hierarchical properties were further investigated for class image segmentation in [136]. The authors improved results of conditional random fields (CRF) tree-based methods by adapting potential functions to image data that is made up of texture features, shape and intensity. This approach is similar to our multiscale strategy in Chapter 4 that includes observed data factors separate from pairwise potentials, a move which is in contrast to ambiguous MRF simplifications.

\(^6\)A Bayesian metanetwork is a collection of associated Bayesian networks with a defined relation that ensures probability distributions associated with nodes of every previous network depend on distributions related with nodes of the next network.
2.4 Relation to Neural Networks

Graphical models are related to neural networks through Boltzmann machines that solve energy functions using stochastic dynamics. A Boltzmann machine [81] is a Markov network for binary variables whose distribution (see Eq. (2.1)) is defined in parametric form as

\[ P(x) = \frac{1}{Z} \exp \left( \sum_{u<v} w_{u,v} x_u x_v - \sum_u b_u x_u \right), \]

(2.30)

where the parameters \( w_{u,v} \) and \( b_u \) represent the interaction strength and bias respectively. It can be regarded as an MRF, as is evident from the definition of its Gibbs distribution in Equation (2.30). As mentioned earlier, variational approximations are the popular methods for inference and learning, often using Gibbs sampling. To this end, a Boltzmann machine that involves stochastic processing is also understood to be a particular form of a neural network [3, 80]. Moreover, the energy function from the Boltzmann machine (2.30) equates to the expression of the energy function for output states of the neural networks introduced by Hopfield [83], i.e.

\[ E(x) = \sum_{u,v} w_{u,v} x_u x_v - \sum_u b_u x_u, \]

where the weights \( w_{u,v} \) are now commutative. The neural networks with this type of energy function are called recurrent neural networks.

We provide further illustration of these concepts in modelling terms. To start off, consider the case when the variable set \( V \) contains both hidden \( h \) and observed variables \( x_{\backslash h} \), for which the variable set is said to be incomplete. The two distinct variable sets are referred to as layers, and if variables within the same layer do not interact with each other, that specific class of Boltzmann machines is formally known as restricted Boltzmann machines (RBM) [163, 54]. On the other hand, if there are interactions between variables in one layer, then the Boltzmann machine is a more general MRF. In a neural network analogy, the nodes and edges of an RBM equate to neurons and sensory connections between neurons respectively. We can then alternatively write the formulation in Equation (2.4) for only the
observed variables $x \setminus h$ in terms of the two parts, i.e.

$$P(x \setminus h) = \sum_h P(x \setminus h, h) = \sum_h \frac{1}{Z} \exp \left( -E(x \setminus h, h) \right).$$

(2.31)

Interpreting an RBM as a stochastic neural network means that inference for the RBM given by Equation (2.31) involves converting the problem into an optimisation setting. The joint distribution $P(x \setminus h)$ can be approximated with a simpler distribution $Q(h)$ to initiate the optimisation process. The optimisation is based on minimising the distance between the unknown distribution $Q(h)$ and $P(x \setminus h)$ in a way that estimates the free energy associated with $Q(h)$.

This variational treatment of Boltzmann machines as neural networks is well documented in the texts [82, 16, 24, 95]. In particular, Hinton and van Camp [82] fit parameters of the neural network to data by using an approximate distribution $Q$. Using this distribution contrasts the MCMC sampling approach that we have mentioned in Section 2.2.1.2. Their approach infers both variables and parameters. The method is called variational Bayes EM, as the posterior is inferred in an E-step and the parameters are determined in an M-step. Furthermore, in the E-step an ensemble of parameters are optimised to obtain their expected value, in contrast to just taking their most probable estimate. However, it is important to note that the descriptions for the two steps depend more on the problem in hand. In this thesis, we will later consider taking a geometric mean to MAP estimates from adjacent layers and combine evidence to improve results in computer vision tasks. Meanwhile, Jordan et al. [95] and their rich reference list provide contexts in which variational methods are applicable to neural networks through the variational Bayes EM method discussed in this section. The work also relates this method to the mean field methodology, since its modelling hinges on approximating the posterior $Q$ with a factorisation of both the parameter distribution $Q(\theta)$ and of the variables $x$, i.e.

$$Q(x; \theta) = Q(\theta)Q(x) = Q(\theta) \prod_i Q_i(x_i),$$

where $\theta$ denotes the parameters. Recall that we have briefly examined the theory behind the mean field method in Section 2.2.1.2 above.
Figure 2.7: Factor graph examples from the main applications listed in Section 2.5. Left: Depiction of a factor graph from decoding. Middle: Forney-style factor graph encountered in wireless networks [186]. Right: Factor graph from biological systems [176].

2.5 Applications

Factor graphs originated in digital communications and signal processing but have become powerful tools in various other areas. They are now mainly found in statistical applications that have many random variables interacting with each other in complex ways. This is mainly because of the rise in computational capabilities, which have greatly increased the possibility to implement them. Further, as factor graphs generalise graphical models, they are applicable to fields which have graphical models as their building blocks.

The proliferation of factor graphs is also largely due to their association with propagation algorithms, mainly the sum-product algorithm. This is because it can be shown that most algorithms applied to other graphical models, such as the Viterbi [55] for decoding, probability propagation [138, 111, 158] in Bayesian networks, the Kalman filters [9] on trees, and even the fast Fourier transform [6] on Abelian groups, are special cases of the sum-product algorithm.

In the following, we will review some of the main areas in which factor graphs have been applied, namely communications, wireless networks and biological systems. We have provided the example factor graph visualisations from these applications in Figure 2.7 according to the same order.
2.5.1 Decoding

The main task to which factor graphs are applied in decoding is deducing the best estimate to a message that has been transmitted through a noisy channel. The history of applying graphical models to this problem began in 1963 with the Gallager algorithm [61]. The codes encountered in their communication systems alluded to a matrix representation of the codes\(^7\), which in turn was used as an incidence matrix between the factors (rows) and variables (columns) of a factor graph. To this end, any linear code \(C\) is defined in terms of the relationship between its elements, i.e. codewords \(\mathbf{x} = [x_1, x_2, \ldots, x_n]\), and the parity-check matrix \(\mathbf{H}\) as

\[
\mathbf{x} \in C \iff \mathbf{H} \mathbf{x}^T = \mathbf{0}.
\]

Figure 2.7(a) shows the factor graph from an example parity-check matrix, where its left side has the variables, and the right side are the factors from the rows of the matrix. In the Gallager decoding algorithm, messages are also probabilities which here represent the belief measure on the value of a codeword bit \(x_i\). As is the case in this thesis, the computations involve using sum and product operations which, in the decoding algorithm, are carried-out using corresponding message passing equations called parity-check equations. These equations are therefore interested in computing the a-posteriori probability using the likelihood ratio \(l\) to decode the value of a bit \(x_i\) given the received word \(y\), where the ratio is defined as

\[
l(x_i) = \frac{P(x_i = 0 | y)}{P(x_i = 1 | y)}.
\]

(2.32)

Notice that here, the likelihood ratio is expressed for each bit \(x_i\) from the parity-check matrix \(\mathbf{H}\). The parity-check equations then involve computing and looking-up bit values to determine codeword estimates \(\hat{x}\). In addition, every decoding sequence involves checking for the condition \(\mathbf{H} \hat{x}^T = \mathbf{0}\).

Therefore, in the real-world application the amount of information to be trans-

\(^7\)The matrix is referred to as the parity-check matrix \(\mathbf{H}\), in the community. For the example in Figure 2.7(a),

\[
\mathbf{H} = \begin{bmatrix}
1 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 1
\end{bmatrix}
\]
mitted made the dimension of the parity-check matrix large, relative to the computational power available at that time. In addition, the discussion above highlights that the algorithm itself involved numerous multiplications that made it costly to implement with the computational power available to Gallager. As a result, computational capability limitations at the time meant the algorithm was not fully utilised. This was a blessing in disguise, as algorithms such as the Viterbi, the BCJR, and others came about. Moreover, this shift in focus resulted in the discovery of turbo codes which allowed any amount of information under the so-called Shannon capacity to be transmitted. It was only in the mid-90s that Wiberg et al. used improved graphical representations to demonstrate that most of the then-existing algorithms for turbo codes were special cases of Gallager’s decoding algorithm. This meant that algorithms for turbo codes and decoding began to be viewed as the same. Meanwhile, computers had become faster and more powerful and researchers could revisit the Gallager algorithm with new insight and exceed expectations by making use of turbo codes. Recent developments, including variants of the sum-product that perform better when a factor graph with larger cycle-length is constructed, can be found in . This is since a given code can be represented by different parity-check matrices, which in turn yield different factor graphs and subsequently, girth. At the same time, sum-product decoding is affected by the chosen method to recover codes, for example, the stability of computations can affect the decision of whether to compute the a-posteriori probability directly, or to compute the a-posteriori probability through the likelihood ratio . These questions, therefore, raise the research challenge to determine which variant of the sum-product best suits a particular graph - as determined by the chosen parity-check matrix - to achieve optimal results.

8 Gallager used an IBM 7090 computer which required many hours for computation.
9 See footnote 8.
10 The girth of a graph is the minimum cycle-length of the graph.
2.5.2 Wireless Networks

The advancement in wireless communications has seen increased installation of more and more sensor nodes, creating sizeable wireless sensor networks. This has, in turn, presented monitoring and resource challenges. Factor graphs have emerged as a crucial tool in supervising network performance through determining and measuring network parameters \[ 123 \] to help in this endeavour. The problem is concerned with mapping the path data flows within the network by determining the status of each link or edge. The variable nodes are the entities associated with data - either computers, routers or mobile phones - that send data packets to each other. The factors are parameterised by a Bernoulli distribution whose random variable is the state of the link. Another set of factors holds the conditional distribution of the path data given the links, and is expressed as a factorisation of Dirac measures. The work in \[ 123 \] profits from the explicit factorisations and the concurrent message scheduling in independent layers of their factor graph to reduce the complexity of the problem. In the field of wireless networks, this problem is referred to as network inference \[ 34, 39 \].

Another recent use of factor graphs in wireless networks is for location awareness of sensors \[ 186 \]. Additional information about the location of a sensor in a network enhances a network’s capability for applications such as emergency services, security and health-care monitoring. To this end, the “state” of a node in \[ 186 \] refers to several properties, including its reference position, velocity and orientation. Wymeersch et al. \[ 186 \] provide a reference list of example applications and propose a technique for finding the location of sensors in a dynamic network. The dynamic nodes follow a memoryless walk and are independent, such that the factor graph includes factors with transition probabilities of variable nodes across different states over-time. Other factors account for the state information of adjacent nodes to compute the conditional local likelihood of states, given the information in the network. The sum-product algorithm over the wireless network (SPAWN), then computes the MAP estimate to determine node locations. It is important to note that this work adopts the Forney-style factor graphs of \[ 118 \] to solve a Bayesian formulation of the localisation task. In Figure \[ 2.7(b) \].
we show a Forney-style factor graph from [186] for the factorisation,

\[ P(x|\tau) = \psi_1(x_1)\psi_2(x_1, x_2)\psi_3(x_1, x_2)\psi_4(x_2, x_3). \]

To create the graph, first, each potential is assigned a vertex. Then, every variable is made into an edge and connected to a vertex if it occurs in the potential at that vertex. If a variable is found in more than two potentials, a “=” vertex is created and dummy variables, i.e. edges, are connected between this vertex and all the corresponding potential factors where the variable appears. The authors in [186] provide a detailed description on how to derive this illustration. A more efficient version of the algorithm involving a reduced number of messages was presented in [2]. Other implementations of factor graphs in the field of wireless networks include network control [35], distributed inference [187], clock synchronisation [5] and more importantly, security for next-generation networks [116].

2.5.3 Biological Systems

Recently, the factor graph framework has also been introduced to biological systems to help integrate gene databases into biological models. Gat-Viks et al. [62] established a general probabilistic model that connects biological entities such as mRNAs, proteins and metabolites. The conditional probabilities in the factor graph are regulation function priors which convey the likelihood that the biological variables attain certain states given their neighbourhood. They demonstrated the utility of this general factor graph formulation to learning problems that incorporated both continuous and discrete measurements, and to infer the activity of proteins given gene expression data. In addition, they also introduced discretiser distributions, which express the joint distribution between the discrete and continuous measurements, to the probabilistic factor graph formalism. Around the same time, factor graphs were also brought into the computational biology community to model a gene transcription regulation system [191]. The generative model in [59] analysed vast protein-coding datasets to verify familiar proteins, and discovered that there may be many gene proteins still to be identified in existing DNA databases.

A similar application of factor graphs to biological systems is in predicting
the most likely interactions, and subsequently the resulting genetic network when cancerous cells invade healthy tissue [176]. In order to understand the fundamental events behind the evolution of normal cells into cancer cells, i.e. carcinogenesis, they consider a network setting in which the interaction of signaling genes (S-genes) is predicted by effect genes (E-genes). They model the real-world, genetic interaction phenomena by improving the nested effects model. The nested effects model clusters genes based on the relations of their observed responses after perturbing them. With the help of factors in the graph, the set of possible relationships between S-genes effected by the E-genes is expanded. The biological system is illustrated in Figure 2.7(c) where different sets of factors are involved. The factors represent the different conditional probability terms in their MAP function. That is, expression factors model the likelihood of expression changes, interaction factors model the connection and interaction between E-gene states and S-genes, and transitivity factors represent interaction priors that enforce consistency in the observed changes. Of notable interest are the transitivity factors $\tau$, which are higher order priors that enforce smoothness on the pairwise S-gene interaction variables. Using the factor graph framework generalises and increases the accuracy of the nested effects model.

2.5.4 Other

Apart from the main applications discussed above, factor graphs have also proven to be vital in other domains. For instance, in natural language processing, they have been used to improve on CRFs to model “skip-chain” dependencies [42], in economics for house-price prediction [37], and in signal processing [23, 118, 165] and robot navigation systems [89, 36] for detection and information fusion.
3.1 Introduction

In this chapter, we explore the max-product message passing of factor graphs for inference tasks which can be graphically represented in terms of interconnected variables realised in a layered arrangement. We note that, structured in this manner, the maximum a posteriori (MAP) inference on the graph can be tackled by rearranging factor nodes into subtrees. We do this by using a Delaunay triangulation and the maximum spanning tree algorithm so as to obtain subtrees of low treewidth. Further, we exploit the fact that in a tree, joint inference can be attained efficiently via the max-product algorithm. The method presented here is quite general in nature and can be applied to a variety of problems defined over interdependent variables. We illustrate its utility for image segmentation and defogging. We also compare our results with those yielded by other techniques elsewhere in literature.

3.2 Background

As mentioned earlier, many inference problems in graphical models can be viewed as being comprised of multiple interrelated sub-problems with individual constraints. These sub-problems can be considered to be individual systems that are interacting with one another through complex relationships. In a Bayesian framework, these problems have mainly been modelled using Markov random fields (MRFs). This is because several applications can be modelled in a way that makes use of MRFs representing either at different scales [121], resolutions [114] or as a time series [13]. In these approaches, each resolution, scale or time
is often assigned its own MRF.

Multilayered MRFs have been applied to segmentation \cite{99, 92, 114}, motion estimation \cite{13, 182} and tracking \cite{2}. For clustering, each layer corresponds to a segmentation map derived from different features \cite{99}, label fields \cite{92} or scales \cite{114}. For motion analysis, a video sequence is considered to be a series of images, each of which corresponds to a layer in the graphical model. In a related approach, Ablavsky and Sclaroff \cite{2} have used a layered MRF so as to track partially occluded objects. In \cite{2}, the layers are formulated with particular emphasis on representing occluders using mobility and visibility constraints. These “occluder layers” are ordered according to their distance with respect to the camera centre and interact by sharing observations of the tracked object through activity zones.

In contrast with the approaches above, coupled and factorial Markov random fields consider two or more sets of variables which correspond to separate MRFs and model their interconnections between layers more explicitly. In coupled MRFs (CMRFs), the variables are estimated simultaneously through a layer-wise alternating maximisation procedure between layers. These models can be traced back to the work in \cite{124}, where the interlinked CMRF consists of one binary valued MRF for both edges and image intensities. The coupled MRF has been employed for line processes \cite{64, 124}, image restoration \cite{172} and optical flow \cite{79}. Further, coupled MRFs based on contrasting assumptions can be found in segmentation \cite{87, 188} and tracking \cite{189}. Huang \textit{et al.} \cite{87} integrate deformations and region-wise configurations to carry out segmentation. In \cite{87}, regions are marked by closed contours, whereby the probability of an image pixel being in a region is dependant on its distance from a contour. Xia \textit{et al.} \cite{188} proposed a segmentation method based upon an MRF that models the labelling problem in hand making use of a coupled MRF which employs colour and texture features. For multiple target tracking, Xue \textit{et al.} \cite{189} use three MRFs coupled with each other so as to capture the probabilities for the targets, their occlusion and the association between them. In a similar fashion, Narashima \cite{132} performs stereo matching by employing a coupled MRF to model the interactions between object boundaries and their surface normals.

In factorial MRFs \cite{100}, the layered structure of the graph is exploited so
Factor graphs \[ \text{as to effect inference based upon an alternating expectation-maximisation (EM) algorithm. In} \] \[ \text{factorial MRFs are used for defogging/dehazing, where} \] \[ \text{the scene depth map and albedo represent two layers. In this manner, the depth} \] \[ \text{and albedo variables each describe an independent MRF where the “complementary” variables} \] \[ \text{are viewed as observables. In} \] \[ \text{closed form solutions for defogging using factorial MRFs have been presented. These make use of a relaxed} \] \[ \text{formulation of the problem in hand} \] \[ \text{or non-linear optimisers} \].

Note that, conceptually, the fundamental difference between coupled MRFs, factorial MRFs and multilayer MRFs is subtle. In this chapter we present a method for inference in factor graphs for tasks that can be graphically represented in terms of interconnected variables in a layered arrangement. This treatment naturally leads to maximum a posteriori inference using max-product message passing. Thus, this chapter not only provides a means for a tractable way to perform inference using local representations, but also allows for factorial, coupled and multilayer MRFs to be viewed as relational structures whose between and intra layer variable relationships are governed by the factors in our graph.

The rest of this chapter is organised as follows. We commence by introducing factor graphs and their layered structure in Section 3.3. In Section 3.4, we show how a local exact solution for the subtrees constructed using Delaunay triangulations can be achieved by applying the max-product algorithm. We do this by taking advantage of the explicit factorisation of factors in factor graphs. We elaborate further on the computation of the Delaunay triangulation and the implementation of our approach in Section 3.5. In Section 3.6, we illustrate the utility of our method in two applications which have been tackled elsewhere using layered MRFs, i.e. segmentation and image dehazing. Finally, we conclude on the work presented here in Section 3.7.

3.3 Factor Graphs

Factor graphs \[ \text{are relational structures such that multivariate or probability mass functions can be factored accordingly. Thus, they are a general form of directed and undirected graphical models consisting of two sets of nodes, i.e. variables and factors. In the case of probability mass functions, factor graphs} \]
capture the relationships between variables through conditional probability distributions defined in factor nodes. Thus, a factor graph is formally defined as a graph $G = (N, E)$ comprising of a node set $N$ and an edge set $E$. The node set can be further decomposed into two subsets, one for variables $V$ and another for factors $F$, such that $N = V \cup F$.

Recall that, in factor graphs, the factor node moderates the influence of surrounding variables via the local conditional probability distribution associated to it. On the other hand, the edge set connects variables to factors, thereby parameterising the conditional probabilities defined by the graph. Here, we view this composite function of the graph as a factorisation of the Gibbs distribution $P(x)$ over the set of variables $x$ given by

$$P(x) = \frac{1}{Z} \prod_{c \in C} \psi(x_c),$$

where $Z$ acts as a normalisation constant and we have defined $P(x)$ in terms of the potential functions $\psi(x_c)$ at each of the nodes $x_c$ corresponding to the subset of variables comprising the clique $c$. This can also be viewed as the Gibbs measure corresponding to a Boltzmann distribution [101] over the variable nodes in the graph.

In Figure 3.1, we show a typical factor graph whose variables are given by round nodes. Note that, from the bipartite node-representation in the figure, we
can better appreciate the structure of the Gibbs parameterisation used throughout the chapter. Our choice stems from the fact that such a treatment naturally captures the conditional independence between nodes by making explicit the mathematical relationships between variables as induced by the factor nodes \[25, 103\]. We also note that, for the sake of efficiency, it is preferable to organise and express these relationships by considering subsets of connected variables as defined by the cliques in the graph. This is as the variables in a clique define a local conditional or joint probability as factored from the graph’s probability mass functions. Note that, as illustrated in Figure 3.1, cliques are comprised of the variables in the factors.

In practice, the left-hand side of Equation (3.1) is a joint distribution over the observables \(\tau\), i.e. image features, pixels, superpixels, etc., and the state of variables \(x\), such that

\[
P(x) \propto P(x, \tau). \tag{3.2}
\]

This can be simplified to yield

\[
P(x, \tau) = P(x|\tau)P(\tau), \tag{3.3}
\]

where

\[
P(x|\tau) = \frac{1}{Z} \prod_{c \in C} \psi(x_c; \tau_c), \tag{3.4}
\]

and \(P(\tau)\) is a constant defined over the set of observables.

### 3.3.1 Layered Representation

As mentioned earlier, a number of practical inference problems can be viewed as interrelated graphical structures arranged in a “layered” fashion. In such cases, the nature of the conditional distributions expressed by the factorisation in Equation (3.4) yields

\[
P(x|\tau) = \frac{1}{Z} \prod_{c \in C} \psi(x_c; \tau_c)
\]

\[
= \frac{1}{Z} \prod_u P(\tau(u)|x(u)) \prod_\gamma P(x_\gamma(u)). \tag{3.5}
\]
The second line in Equation (3.5) above follows from applying Bayes’ rule. As a result, the observable $\tau(u)$ becomes a vertex in the graph $G$ on the layer indexed $\gamma$, such that $x_{\gamma}(u)$ denotes the vertex attributes. The variables interacting between layers are represented by

$$x(u) = \bigcup_{\gamma \in \mathcal{L}} x_{\gamma}(u)$$

in Equation (3.5). Notice that $Z$ corresponds to the marginalisation of the numerator in Equation (3.5). This is equivalent to the marginal distribution $P(\tau)$ over the entire graph.

We illustrate the description above in Figure 3.2, where a set of superpixels in an image characterise two sets of connected image tokens that correspond to variables in the graph. In the figure, these are linked through factors. Note that, from this arrangement, the layered organisation of the graph becomes evident.

From inspection, we can appreciate that Equation (3.5) suggests layers have an \textit{a priori} uncoupling which is indicated by the factorisation

$$P(x) = \prod_{\gamma \in \mathcal{L}} \prod_{u} P(x_{\gamma}(u)).$$  \hspace{1cm} (3.6)

This is consistent with the intuition that observables, \textit{i.e.} $\tau$, can be approximated
by a set of random variables across different layers.

These variables exchange information and interact with each other depending on the underlying model governing the observables themselves. Further, the likelihood term of these interacting variables is determined by the conditional distribution

$$P(\tau(u) | x(u)) = \prod_{\gamma \in L} \sum_{\phi \sim \gamma} P(\tau(u) | b(\phi(u)), x_\gamma(u)) P(b(\phi(u)) | x_\gamma(u))$$

which follows the notion that the probability of observing a particular state of $x_\gamma(u)$ for other layers is dependent on the state belief vector $b(\phi(u))$ from other layers $\phi$. The notation $\phi \sim \gamma$ indicates that layer $\phi$ is adjacent to layer $\gamma$.

In Equation (3.7), the contribution of the $\gamma^{th}$ layer to the probability $P(\tau(u) | x(u))$ is governed by the state of $x_\gamma(u)$ given through the categorical distribution

$$P(b(\phi(u)) | x_\gamma(u)) = \begin{cases} \prod_{k=1}^{\phi|} P(b(\phi(u))_k, x_\gamma(u)), & \text{if } x_\phi(u) = \varsigma \\ 0, & \text{otherwise} \end{cases}$$

where $b(\phi(u))_k$ symbolises that the variable $x_\phi(u)$ is believed to be in state $k$ given the state of variable $x_\gamma(u)$, and $|\phi|$ are the number of states in layer $\phi$. In Equation (3.8), we have also denoted the value achieved by the variable $x_\phi(u)$ at state $k$ as $\varsigma$. The likelihood of an observation in a layer $\gamma$ is, hence, estimated using its dependency on the beliefs $b(\phi(u))$ originating from the layers $\phi$ adjacent to $\gamma$, i.e. $\phi \sim \gamma$. This observation is relevant to our analysis since the functional in Equation (3.8) implies that interactions between layers can be approximated by the likelihood of variables $x_\phi(u)$ in layers $\phi$ to achieve state $\varsigma$ when associated with the variable $x_\gamma(u)$.

In Figure 3.3, we show the graph corresponding to the formulation in Equation (3.5). In the figure, each variable (round nodes) accounts for an image token at different layers. These can be features, colour, edges, etc. and are accompanied by their factor nodes (squared) representing the corresponding likelihoods and priors. Assuming a connected neighbourhood, an image token is then adjacent
Figure 3.3: Left-hand panel: A factor graph realised in layers; Right-hand panel: Detail of the area outlined on the left-hand panel of the graph showing the beliefs sent to and from variables along the graph edges. Note that in this figure we have used $\gamma'$ to represent another layer different from $\gamma$ and $\phi$.

to its neighbours through these factors, which yields the local structure on the left-hand side of Figure 3.3.

3.4 Inference on Layered Graphs

We now turn our attention to the Gibbs distributions as factorised over the clique potentials given in Equation (3.5). In order to gain a better insight into the layered representation presented in the previous section, we note that once a series of layers are connected to each other, a subset of inter-layer edges can be obtained via a triangulation. This allows for the construction of a structure connecting variables in different layers so that factors can be composed using the conditional distributions in Equation (3.5).

3.4.1 Max-product Message Passing

Recall that our aim of computation in the inference process in hand is to recover the marginal probabilities $P(x_{\gamma}(u))$ for a particular variable or set of variables at
the factors. This is often done using a sum-product \cite{108} or belief propagation algorithm \cite{192}. Here, however, we use the max-product algorithm. Our choice is motivated by the notion that, in the max-product, the messages between factors and variables are not an approximation to the marginal distribution but rather a "scoring" function that can be used to compute a posterior probability \cite{108}.

This permits posing the inference problem in Equation (3.5) over the graph $G$ as a maximum a posterior (MAP) probability formulation. Thus, we optimise the target function given by

$$
\arg\max_x P(x|G) = \arg\max_x \frac{1}{Z} \prod_u P(\tau(u)|x(u)) \prod_\gamma P(x_\gamma(u)).
$$

(3.9)

We do this by taking an approach akin to belief propagation, and will make use of Figure 3.3. To this end, let the variable node $x_\gamma$ send a message to a factor node $f$ given by

$$
\mu_{x_\gamma \rightarrow f}(x_\gamma) = \prod_{g \in \mathcal{N}_{x_\gamma}} \mu_{g \rightarrow x_\gamma}(x_\gamma),
$$

(3.10)

where $\mathcal{N}_{x_\gamma}$ is the set of factors neighbouring the variable node $x_\gamma$, regardless of the layer under consideration.

Consider the likelihood factor denoted by $h$. This factor node sends the message $\mu_{h \rightarrow x_\gamma}(x_\gamma)$ to the variable in layer $\gamma$ according to the expression

$$
\mu_{h \rightarrow x_\gamma}(x_\gamma) = \max_{x_c \setminus x_\gamma} \psi_h(x_c) \mu_{x_\phi \rightarrow h}(x_\phi) = \max_{x_c \setminus x_\gamma} P(\tau(u)|b(x_\phi(u)))P(b(x_\phi(u))|x_\gamma(u)) \prod_{g \in \mathcal{N}_{x_\gamma}} \mu_{g \rightarrow x_\gamma}(x_\phi),
$$

(3.11)

where $x_c$ denotes all the variables in the neighbourhood of $x_\gamma(u)$.

Moreover, from Equation (3.11), we can conclude that the variable-to-likelihood factor message is given by

$$
\mu_{x_\gamma \rightarrow h}(x_\gamma) = \prod_{g \in \mathcal{N}_{x_\gamma}} \mu_{g \rightarrow x_\gamma}(x_\gamma).
$$

(3.12)

where $\mu_{g \rightarrow x_\gamma}(x_\gamma)$ is the factor-to-variable message within layers. These factors
depend upon the priors $P(x_{\gamma})$, and can be expressed explicitly as

$$
\mu_{f \rightarrow x_{\gamma}}(x_{\gamma}) = \max_{x_{c} \setminus x_{\gamma}} \psi(x_{c}) \prod_{y \in x_{c}, y \neq x_{\gamma}(u)} \mu_{y \rightarrow f}(y) \prod_{y \in x_{c}, y \neq x_{\gamma}(u)} P(x_{\gamma}(u), x_{\gamma}(v)) \mu_{y \rightarrow f}(y). \quad (3.13)
$$

Note that in Equation (3.13) we have expressed the prior probability as a pairwise potential $P(x_{\gamma}(u), x_{\gamma}(v))$. To take our analysis further, we note that the max-product algorithm computes exact probability marginals of variables in a tree [138, 6]. It can also be applied to general factor graphs, such as lattice or grid-like structures. This contrasts with other methods such as loopy belief propagation, which sometimes deliver poor estimates in graphs with cycles [192]. To profit from this trait, here we express factor graphs using local tree structures. To this end, we use a Delaunay triangulation to define constraints on the variables and create adjacency relationships in which the variables become vertices of simplexes.

These combinations are the adjacency relationships that, for instance, in image segmentation, capture the neighbourhood structure induced by the spatial arrangement of pixels in the image. Furthermore, we divide or partition the graph obtained from these adjacency relations and employ a variable elimination strategy to the subgraphs to ensure the triangulation is complete. This engenders a collection of cliques, which are linked together through a maximum spanning tree on the cliques. This, effectively, yields a junction tree.

Our motivation for using a Delaunay triangulation and local junction trees stems from the fact that their treewidth\(^1\) is low. This ensures an efficient implementation of our message passing algorithm, as it reduces the number of possible joint combinations of variables in a clique. To compute a local junction tree, we depart from a Delaunay triangulation and recursively bipartition the graph until a pre-set treewidth is achieved. To do this, at each bipartition step, we apply the normalised cut [161] making use of the similarity between adjacent variables in the graph. These local partitions for each layer are then used to compute the

\(^1\)The treewidth of a graph is the number of variables in its largest clique
§3.4  Inference on Layered Graphs

Figure 3.4: From left-to-right: Delaunay triangulation over a set of image regions, graph partition yielded by the normalized cuts algorithm and resulting junction tree computed from the sub-graph in the middle panel.

global MAP approximation for Equation (3.9).

In Figure 3.4(b) we illustrate the procedure above when applied to a set of image regions. In the left-hand panel, we show the adjacency relationships between region centroids obtained when computing a Delaunay triangulation. In the left-hand panel we have also shown the graph cut obtained by the method in [161] and the resulting subgraph in the middle panel. The final junction tree is shown in the right-hand side of Figure 3.4(c). The junction tree shown in the figure has been obtained by applying the maximum spanning tree computation algorithm in [142]. Note that in, the figure, and for the sake of clarity, we have omitted the factors on the graphs and used a single layer.

In algorithm 1 we show the step sequence for our inference process. Note that the strategy described above is quite general, whereby the graph bipartition can be performed by making use of other graph cut algorithms elsewhere in the literature. Furthermore, if a partition set \( \mathcal{P} \) is in hand, the Delaunay triangulation can be applied to individual subgraphs. This follows the intuition that layers may be partitioned so as to enforce hard constraints on the topology of the factor graph, with these partitions linked together in an application dependant manner. Thus, in Algorithm 1 we require a partition set at input.
Algorithm 1 Main Routine

Require: Factor graph $\mathcal{G}$, initial estimate for the likelihood $b(x_\gamma(u))$ at each of the $l$ layers

Require: Partition set $\mathcal{P}$

Ensure: MAP estimate and beliefs $b(x_\gamma(u))$ for all $\gamma$ and tokens $\tau(u)$

1: for all $\gamma \in \mathcal{L}$ do
2:   for all $p \in \mathcal{P}$ do
3:     Connect variables in partition $p$ by a Delaunay triangulation.
4:     Use connectivity to compute the potentials for the factors $f$.
5:     Use Delaunay triangulation to recover the cliques $c_i$.
6:     Use the maximum spanning tree algorithm in [142] to link cliques and obtain rooted tree $R$.
7:     Run Algorithm 2 on $R$.
8:     Compute the partition beliefs $b_p(x_\gamma(u))$ by marginalising over clique beliefs $B(x_{\gamma,c_i})$. These beliefs will be used for the adjacent layer computations using Algorithm 2 for each layer.
9:   end for
10: end for
11: return Beliefs $b(x(u))$

3.4.2 Inference in Local Junction Trees

We now elaborate further on the message passing scheme for our local junction trees of clique factors. Here, we will assume the root node is in hand. Note that this is necessary since the operations of the max-product algorithm involve two steps. The first of these steps is concerned with passing messages to the root node. The second stage involves passing on messages from the root node back to the leaf nodes.

Note that the equations presented in Section 3.4.1 can be combined so that the max-product scheme is performed over the cliques. The message sent back and forth between two cliques $c_i$ and $c_j$ in this case becomes

$$\mu_{c_i \rightarrow c_j}(s_{i,j}) = \max_{c_i \setminus \{s_{i,j}\}} \Psi(x_{c_i}) \prod_{k \in (\mathcal{N}_i \setminus \{j\})} \mu_{c_k \rightarrow c_i}(s_{k,i}), \quad (3.14)$$

where $\Psi(x_{c_i})$ is the initial potential as defined in Equation (3.28) and $\mu_{c_k \rightarrow c_i}(s_{k,j})$ are the messages incoming to the clique $c_i$. The variables in the intersection of the cliques $c_i$ and $c_j$ are known as the separator set $s_{i,j}$ and $\mathcal{N}_i$ denotes the cliques in the neighbourhood of clique $c_i$. 
§3.4  Inference on Layered Graphs

Here, each clique factor consists of factors within the layer and across layers for each of the variables. If we consider the factor nodes to be leaf nodes, the message sent within the layer to the respective variables is

$$\mu_{h \rightarrow x_\gamma}(x_\gamma) = \max_{x_c \setminus x_\gamma} \psi(x_c = \{x_\gamma, x_\phi\})$$

$$= \max_{x_c \setminus x_\gamma} P(\tau(u)|b(x_\phi(u)))P(b(x_\phi(u)|x_\gamma(u))) \quad (3.15)$$

Similarly, the message from the within-layer factors is

$$\mu_{f \rightarrow x_\gamma}(x_\gamma) = \max_{x_c \setminus x_\gamma} \psi(x_c = \{x_\gamma(u), x_\gamma(v)\})$$

$$= \max_{x_c \setminus x_\gamma} P(x_\gamma(u), x_\gamma(v)). \quad (3.16)$$

Let $\Psi(x_c)$ be the potential in Equation (3.14). If this potential is computed for all cliques before the message passing procedure begins, the factors are then governed by either of the messages in Equations (3.15) or (3.16). Subsequent messages from variables to factors are defined by Equations (3.10) and (3.12). We can provide an intuitive interpretation to this process by viewing the messages in Equation (3.14) between cliques as amalgamations of the potentials in local messages. This is also consistent with Equations (3.10), (3.11), (3.12) and (3.13). Viewed in this manner, the clique potential is a conditional distribution for all possible permutations of the variables in the clique.

In Algorithm 2, we show the step-sequence of the max-product scheme presented here. In the algorithm, our aim of computation is the updated clique potentials. This is as the objective of solving the MAP problem in Equation (3.9) can be viewed as that of recovering the most likely state of all variables in the graph. Moreover, this also allows for the computation of the beliefs $B(x_{\gamma,c_i})$ for any clique $c_i$ given by

$$B(x_{\gamma,c_i}) = \Psi(x_{\gamma,c_i}) \prod_{k \in N_i} \mu_{c_k \rightarrow c_i}(s_{k,i}). \quad (3.17)$$

In Equation (3.17) we have made explicit that the variables of the clique $x_{\gamma,c_i}$ are in layer $\gamma$. As a result, the marginal distribution or belief $b(x_\gamma(u))$ for a particular variable $x_\gamma(u)$ is obtained via marginalisation over all other variables
Efficient Inference in Factor Graphs

Algorithm 2 - Junction tree

Require: Root node \( r \) and scheduling of the edges \( e \) in the rooted tree \( R \)
Ensure: Every edge \( e \) in \( R \) has messages sent in either direction and \( B(x_c) \)

1: **Comment:** Message passing towards \( r \):
2: for all \( e \) linking cliques \( c_i \) and \( c_j \) of \( R \) do
3:   if \( \Psi(x_{c_{i/j}}) \) unknown then
4:     Compute \( \Psi(x_{c_{i/j}}) \)
5:   end if
6:   Compute \( \mu_{c_i ightarrow c_j}(s_{i,j}) \) (3.14).
7: end for

8: **Comment:** Message passing towards the tree leaves:
9: for all \( e \) between \( c_j \) and \( c_i \) of \( R \) do
10:   Compute \( \mu_{c_j ightarrow c_i}(s_{j,i}) \) (3.14).
11: end for
12: return Updated clique potentials

in a clique that contain \( x_\gamma(u) \). That is

\[
P(x_\gamma(u)) = b(x_\gamma(u)) = \max_{x_{\gamma,c} \setminus x_\gamma} B(x_{\gamma,c}). \quad (3.18)
\]

3.5 Discussion

In this section, we elaborate further on the Delaunay triangulation framework for enforcing adjacency constraints on the graph. We also provide further discussion on the link between our approach and factorial Markov random fields.

3.5.1 Enforcing Adjacency Constraints on Delaunay Triangulations

To enforce adjacency relationships on the factor graph variables over our Delaunay triangulation, we first use Prim’s algorithm \([142]\) to order tokens according to a minimum spanning tree. The algorithm aims to pick a subset of edges that include all vertices, i.e. variables, from the factor graph. To do this, we begin by creating an ordered empty set or list that will contain variables included in the triangulation and choose a root node at random. The variable at this node
Algorithm 3 Factor graph Delaunay triangulation

Require: Set of variable nodes and root variable node $x_r(u)$
Ensure: Every variable $x(u)$ is in the graph $G$

1: Two sets of variables nodes:
   $G_v$, if $x(u)$ has been included into a triangulation;
   $V$, if $x(u)$ has not yet been included.
2: Initialise: $G_v = \{x_r(u)\}$, $V = \{X \backslash x_r(u)\}$.
3: while $V \neq \emptyset$ do
   4: for all $x(u) \in V$ do
      5: Find $x(u) \leftarrow \arg \min_x d(x_u, x_r)$, where $d(\cdot)$ is the Euclidean distance between $x_u$ and $x_r$
   6: end for
   7: Insert $x(u)$ by modifying $G_v$ according to [28].
   8: $V \leftarrow V \backslash x(u)$
9: end while
10: return Triangulated graph $G$

becomes the first element of the list.

Afterwards, we ensure neighbourhood consistency between variables by considering the dissimilarity, i.e. the distance, between variables included in the list and the variables from the graph that are not yet included. We then identify the variable from the excluded set that is closest to one variable in the included set, and record its index in the list. This identify-and-list operation over nearest neighbours is repeated until all variable indexes are recorded. The Delaunay triangulation is finally obtained by adding the ordered points one by one following the approach in [28].

This step sequence is summarised in Algorithm 3. Note that, in Algorithm 3, we have used the Euclidean distance $d(\cdot)$ for locating the set of nearest neighbours. It is worth stressing that the algorithm can be modified in a straightforward manner to use other choices of metric or dissimilarity measurements such as commute times or random walks [70].

3.5.2 Relation to Factorial Markov Random Fields

Now we turn our attention to the relation between our approach and factorial Markov random fields (FMRFs) [100]. To this end, we write the posterior distri-
Efficient Inference in Factor Graphs

The distribution $P(x|\tau)$ in Equation (3.4) as follows

$$P(x|\tau) = \prod_u \prod_\gamma P(\tau(u)|x_\gamma(u))P(x_\gamma(u)). \quad (3.19)$$

Let $x_\gamma(u)$ define a binary random variable. Following the assumption used in FMRFs that the probability of variables $x_\phi(u)$ for adjacent layers is in hand, we can obtain their states and express the marginalisation of the likelihood for layer $\gamma$ as

$$\max_{\phi \sim \gamma} P(\tau(u)|b(x_\phi(u)))P(b(x_\phi(u))|x_\gamma(u))$$
$$\simeq P(\tau(u)|x_\phi^*(u), x_\gamma(u))P(x_\phi^*(u)|x_\gamma(u))$$
$$= P(\tau(u), x_\phi^*(u)|x_\gamma(u))$$
$$\simeq P(\tau(u)|x_\gamma(u)), \quad (3.20)$$

where $x_\phi^*(u)$ denotes the “optimal” state to the hidden variables in layer $\phi$ given the variable $x_\gamma(u)$. The first approximation to the marginalisation of the expression in Equation (3.20) follows from the FMRF assumption described above. In addition, the first two steps of the equation use the product rule of probability, and the notion that the variable $x_\phi^*(u)$ can be considered to be observation since its state is known. Finally, this known state can be absorbed into the token $\tau(u)$, so that the last approximation follows. In Equation (3.20) above, note that we have chosen to be consistent with our earlier developments and employed the max-product algorithm rather than using the max-marginal formulation in [100].

With these ingredients, the distribution $P(\tau(u)|x_\gamma(u))$ defines the likelihood for the variable $x_\gamma(u)$ at $u$ and in layer $\gamma$. This has the advantage that now we can use a set of uncoupled MRFs to approximate the posterior in Equation (3.9). This can be expressed as follows

$$P(x|\tau) \simeq \prod_u \left[ \prod_\gamma P(\tau(u)|x_\gamma(u)) \right] \prod_\gamma P(x_\gamma(u)). \quad (3.21)$$

It is evident that Equations (3.19) and (3.21) are equivalent. Therefore, we can conclude that, if the variables $x_\gamma$ are binary, the condition in Equation (3.20)
simplifies the layered formulation in this chapter to an FMRF. Furthermore, the main change to the message passing stems from an additional intermittent maximisation step at the likelihood factor node so as to estimate $x_{\phi^*}(u)$. This intermittent step corresponds to a Dirac measure, so that having these estimates in hand implies that the message sent in Equation (3.11) becomes

$$
\mu_{h \rightarrow x_\gamma}(x_\gamma) = \delta(x_\gamma - x_{\gamma^*}).
$$

In addition, the belief propagation scheme inside a layer $\gamma$ now employs this distributions to obtain new values of $x_{\gamma}(u)$ for the layer under consideration. Solving for all layers interchangeably in this way corresponds to the expectation-maximisation (EM) procedure for layered MRFs in [100, 173] and factorial hidden Markov models [67]. In the E-step, the posterior probabilities, i.e. the variables at the each location for the layers, are estimated. Meanwhile, the M-step maximises the expected likelihood of the variable states.

### 3.6 Applications

We now illustrate the utility of our factor graph approach to tackle two sample computer vision tasks. These are image segmentation and defogging/dehazing. Our choice stems from the fact that the former is a classical problem which arises in a wide variety of settings whereas one of the methods of choice for the latter is factorial MRFs.

#### 3.6.1 Segmentation

Here, we illustrate the utility of our factor graph for purposes of colour segmentation, where each image consists of a given number of clusters $k = 1, 2, ..., K$. Each cluster is represented by a mean colour value, $\omega_k$, whereby our goal of computation becomes that of assigning each pixel to a cluster, that is, the recovery of the cluster mean colour and the corresponding pixel labels.

Here, we define the probability $P(\tau|x)$ in the MAP problem in Equation (3.9) as a multivariate Gaussian mixture model over the $K$ classes under consideration.
expressed as

$$P(\tau(u)|x(u)) = \sum_k \xi_k P(\tau(u)|x_\gamma(u) = k). \quad (3.23)$$

where $P(\tau(u)|x_\gamma(u) = k)$ is a Gaussian distribution given by

$$P(\tau(u)|x_\gamma(u) = k) = \frac{1}{\sqrt{2\pi|\Sigma_k|}} \exp \left( -\frac{1}{2} (\tau(u) - \bar{\omega}_k)^T \Sigma_k^{-1} (\tau(u) - \bar{\omega}_k) \right) \quad (3.24)$$

and $\bar{\omega}_k, \Sigma_k$ are the mean and covariance for the $k^{th}$ Gaussian in the mixture.

The mixture prior in Equation (3.8) is given as the exponential to a delta function of the similarity of pixels across layers, i.e.

$$\xi_k = P(x_\gamma(u) = k|x_\phi \in N_\gamma)$$

$$= \sum_{N_\gamma} \exp \left( -\vartheta \left( \delta(x_{\gamma,k}(u) - x_{\phi,k}(u)) - 1 \right) \right) \quad (3.25)$$

subject to $\sum_k \xi_k = 1$, where the parameter $\vartheta$ controls the interactions between layers.

Note that characterising likelihoods in this manner provides a means to imposing a smoothness penalty across layers. Thus, the smoothness prior for neighbouring pixels $x_\gamma(u)$ and $x_\gamma(v)$ in a layer $\gamma$, can be expressed as follows

$$P(x_\gamma(u)) = \sum_{v \in N_u} \exp \left( -\alpha (x_\gamma(u) - x_\gamma(v))^2 \right). \quad (3.26)$$

where $\alpha$ is a positive, real constant.

### 3.6.1.1 Implementation issues

In order to construct a multi-layered graph, we obtain a three-layer pyramid by scaling down the image to a half and a fourth of its original size. We then proceed in a similar fashion to that in [11] by first obtaining an over-segmentation of the image using an oriented watershed transform ($owt$). Figure [3.5] shows the images obtained from conducting these steps.

For all pixels in a given $owt$-region, we compute the mean colour and select a
§3.6 Applications

Figure 3.5: Algorithm steps. Left: Input image. Middle left: The input image after rescaling to half its original size and back to full size. Middle right: The image, this time scaled to a quarter of its size and back to full size. Right: Over-segmented image by oriented watershed transform [11], with region centroids in red.

region centroid or superpixel that most likely matches the mean based upon the Euclidean distance, i.e.,

$$\min_{z \in R} \| x(z) - m_R \|,$$  \hspace{1cm} (3.27)

where $z$ denotes pixels in region $R$ and $m_R$ is the mean colour of the region. A region’s superpixel then corresponds to a variable node in the factor graph. We use the spatial position of the region centroids for each scale and compute the Delaunay triangulation on the image plane for each of the three scales at our disposal using Algorithm 3. These triangulations are used for the layer edge-set of our factor graph. The connectivity across layers is obtained by first projecting region centroids for two neighbouring layers onto a reference plane, common to all of the layers (for the finer scale, this would correspond to the image plane). A new Delaunay triangulation for the projected centroids is obtained so that an edge is considered to be across layers if two connected centroids correspond to different scales.

With the factor graph in hand, we obtain the log-likelihood by combining Equations (3.24) and (3.25). This yields the following cost function

$$- \arg\min_{x, \Theta} \log P(x | \tau, \Theta) = \arg\min_{x, \Theta} \log(2\pi)|\Sigma_k| + \sum_u \left[ \frac{1}{2} (\tau(u) - \varpi_k)^T \Sigma_k^{-1} (\tau(u) - \varpi_k) + \vartheta \left( \delta\left(\|x_{\gamma,k}(u) - x_{\phi,k}(u)\|\right) - 1 \right) + \alpha \|x_{\gamma,k}(u) - x_{\gamma,k}(v)\|^2 \right],$$

where $\Theta = \{\varpi, \Sigma\}$ is the set of parameters, as before, $\alpha$ and $\vartheta$ are constants and
$|\Sigma_k|$ is the determinant of the covariance matrix $\Sigma_k$.

For each layer, we apply the normalized cut algorithm \[161\] and connect partitions to each other through the likelihood factors across layers. We then construct local junction trees in each layer and apply the max-product algorithm as presented in Section 3.4.1. Variables connected in each of the triangles or simplexes yielded by the Delaunay triangulation are used to create cliques $\Psi(x_c)$, which combine both likelihood and prior potentials into factors.

Here, we initialise the likelihood distribution to the label most similar to the pixel value using a Kronecker delta distribution. The within layer conditional distributions are used to compute initial potentials using the product of the factors comprising the clique. This yields

$$\Psi(x_{c_i}) = \prod_{j: f_j = \psi(x_c)} f_j, \quad (3.28)$$

Lastly, the cliques are joined by computing their maximum spanning tree using the algorithm in \[15, 103\]. This algorithm recursively links two cliques $c_i$ and $c_j$ if, for all possible edges in the graph, the cardinality of the variables in their intersection, $|c_i \cap c_j|$ is largest. This procedure is conducted until all cliques have at least one edge.

### 3.6.1.2 Results

We now finalise the section by presenting the results yielded by our segmentation method on the BSDS500 dataset \[125\]. We also compare our results with those yielded by other popular segmentation algorithms elsewhere in the literature. To this end, we have used the belief propagation technique in \[50\], normalized cuts \[161\], mean shift \[40\] and loopy belief propagation \[192\].

In Figure 3.6 we show the segmentation results yielded by our method and the alternatives in \[49, 161, 192\] when applied to a number of sample images. Note that our factor graph method yields segmentations that are in good accordance to the natural colour arrangement of the imagery.

We now focus our attention to a quantitative evaluation of our segmentation results. To this end, we have used the boundary-based methodology in \[11\] to
Figure 3.6: Segmentation results. Left-to-right: Sample images from the BSDS500 dataset [125]; Segmentation yielded by the method in [19], Normalised cuts [161], loopy belief propagation [192] and our factor graph approach.
compute a precision-recall plot for our method and the alternatives as applied to the whole dataset. We have done this by computing the gradient of the segmentation image and matching the resulting boundaries to each of the five boundary scales provided by the human ground-truth for the BSDS500 dataset.

In Figure 3.7 we show the precision-recall curves. For the benefit of the reader, we also show the corresponding $F_1$ scores. From the curves, we can appreciate that our approach, the method in [49] and the normalized cuts [161], all outperform loopy belief propagation. Moreover, our approach and the method in [161, 40] yield almost identical $F_1$ scores. Note, however, that our method outperforms the alternatives as recall increases. It is also worth noting that our method outperforms the mean-shift at the recall extrema.

3.6.2 Defogging

We now turn our attention to our second sample application. Removing fog and haze from images is required as a preprocessing step for a wide range of important computer vision tasks such as segmentation, object recognition, face detection, tracking and video surveillance. Existing methods model the dispersion of light
in fog as a linear combination of an exponential increase and decrease in natural light and direct reflected light \[133\].

Thus, here we express the image intensity at a wavelength \(\lambda\) as follows

\[
I(u, \lambda) = L_\infty(\lambda) \rho(u, \lambda) e^{-\beta(\lambda)d(u)} + L_\infty(\lambda)(1 - e^{-\beta(\lambda)d(u)}).
\] (3.29)

where \(d(u)\) denotes the distance between the camera sensor and the object, \(\beta(\cdot)\) is a scattering coefficient, \(\rho(u, \lambda)\) is the pixel albedo and \(L_\infty(\lambda)\) is the power spectrum from the horizon.

Note that the extent to which light is dispersed or scattered depends on the scattering coefficient \(\beta(\lambda) = \frac{k}{\lambda}\) where \(k = \beta(\lambda_{550})\lambda_{550}\) is given by \(\beta(\cdot)\) evaluated at 550\(\text{nm}\). Further, rearranging the equation above yields

\[
\left(1 - \frac{I(u, \lambda)}{L(\lambda)}\right) = e^{-\beta(\lambda)d(u)}(1 - \rho(u, \lambda)).
\] (3.30)

which we can rewrite as follows

\[
\tilde{I}(u, \lambda) = t(u, \lambda)\tilde{\rho}(u, \lambda)
\] (3.31)

using the shorthands \(\tilde{\rho}(u, \lambda) = 1 - \rho(u, \lambda)\), \(t(u, \lambda) = e^{-\beta(\lambda)d(u)}\) and \(\tilde{I}(u, \lambda) = \left(1 - \frac{I(u, \lambda)}{L(\lambda)}\right)\).

If the noise \(\eta\) obtained from the observation \(\tilde{I}(u, \lambda)\) is assumed to follow a normal distribution, then

\[
\eta \sim P(\tilde{I}|\tilde{\rho}, t) = \mathcal{N}(\tilde{I}; \tilde{\rho}t, \sigma^2),
\] (3.32)

where the bold font describes all variables of the same kind, \(i.e.\ t = \bigcup_{u, \lambda} t(u, \lambda)\).

This is an important observation since the joint recovery of the albedo \(\rho(u, \lambda)\) and the transmission \(t(u, \lambda)\) becomes that of maximising the posterior distribution given by

\[
(t^*, \rho^*) = \mathop{\text{argmax}}_{t, \rho} P(\rho, t|\tilde{I}) = \mathop{\text{argmax}}_{t, \rho} P(\tilde{I}|\rho, t)P(\rho)P(t).
\] (3.33)
The inference problem in Equation (3.33) is equivalent to the MAP inference in Equation (3.9) where the distributions \( P(\rho) \) and \( P(t) \) are the independent layer priors for the albedo and depth respectively. Note, however, that our observation likelihood as expressed in Equation (3.7) takes into account the association between albedo and transmission, which is a function of the distance or depth \( d(u) \). As such, the correlation between albedo and transmission is a marginalisation of a uniform distribution of possible permutations of joint probabilities so that Equation (3.8) is now given by

\[
P(b(t(u, \lambda)) | \rho(u, \lambda)) = \sum_t \frac{1}{|P(t(u, \lambda) | \rho(u, \lambda))|} b(t(u, \lambda)),
\]

(3.34)

where \(|P(t(u, \lambda) | \rho(u, \lambda))|\) denotes the number of labels of transmission \( t(u, \lambda) \).

Here, for the prior distributions of albedo and depth, we employ the functions proposed in [135]. For the albedo, the prior is a generalised Gaussian function determined by the image colour gradients. This generalised Gaussian is given by the following exponential power distribution

\[
P(\rho(u, \lambda)) = \prod_{v \in N_u} \exp(-V_{\rho}(u, v, \lambda))
\]

(3.35)

where, the \( v \) variables are constrained to be in the neighbourhood \( N_u \) of \( u \). In the expression above, \( V_{\rho}(u, v, \lambda) \) is a potential function given by

\[
V_{\rho}(u, v, \lambda) = \frac{|\rho(u, \lambda) - \rho(v, \lambda)|^{\psi}}{\eta^{\psi}}.
\]

(3.36)

where \( \psi > 0 \) and \( \eta > 0 \) are the shape and scale parameters, respectively.

Also, recall that, in [135], the albedo distribution is initialised using the expression

\[
P(\rho) = \prod_{\lambda} \prod_u \prod_{v \in N_u} \exp(-V_{\rho}(u, v, \lambda)).
\]

(3.37)

where, as before, \( \rho = \bigcup_{u, \lambda} \rho(u, \lambda) \).

For the depth, note that the transmission distribution is dependent upon the
appearance of the scene. Here, we have followed [135] and initialised the depth distribution using the Laplace distribution. Thus, we have

\[ P(t(u, \lambda)) = \prod_{v \in N_u} \exp(-\mathcal{V}_t(u, v, \lambda)). \]  

(3.38)

where, \( \mathcal{V}_t \) is, again, a potential function which takes the form

\[ \mathcal{V}_t = \frac{|t(u, \lambda) - t(v, \lambda)|}{\varphi}. \]  

(3.39)

Finally, since the prior probabilities involve neighbouring pixels, they can be viewed as instantiations of multiple permutations between labels over the neighbours. As a result, here we employ the Potts model [141] so as to write

\[ P(t(u, \lambda), \rho(u, \lambda), \tilde{I}(u, \lambda)) = \frac{1}{Z} \exp \left( -\frac{\|\tilde{I}(u, \lambda) - \hat{I}(u, \lambda)\|^2}{2\sigma^2} \right) \exp(-\mathcal{V}_\rho) \exp(-\mathcal{V}_t) \]  

(3.40)

where \( \hat{I}(u, \lambda) = t(u, \lambda)\tilde{\rho}(u, \lambda) \) is an approximation to the observation \( \tilde{I}(u, \lambda) \) and \( Z \) is a normalisation constant.

### 3.6.2.1 Implementation issues

For our implementation, we have modelled each colour channel as a factor graph on a grid, with an albedo and depth layer. The albedo label-set consists of 256 intensity values, while we scale the depth comprises 40 different levels. As such, the MAP inference is conducted over these discrete state spaces with each step updating the labels such that the posterior probability is maximised. It is important to note that the max-product algorithm may suffer from floating-point errors as repeated multiplication operations are performed between small probabilities. To delay this occurrence, we employ a log-projection of the max-product, effectively adopting a min-sum algorithm strategy [108].

In our implementation, the inference procedure commences in the albedo layer by using an initial depth estimate computed making use of the variable

\[ d_0(u) = \min_{\lambda} \frac{\log(\tilde{I}(u, \lambda))}{-\beta(\lambda)}. \]  

(3.41)
The motivation for the equation above hinges on the span of the depth at each pixel. Note that the maximum depth at any of the image pixels occurs when \( \rho(u, \lambda) = 0 \), which follows from Equation (3.30). These assumptions were, to our knowledge, introduced in [135]. Here, however, we have introduced wavelength dependence via the scattering coefficient, \( \beta(\lambda) \), making the derivation more general in nature.

Once the initial depth estimate \( d_0(u) \) is in hand, we can directly compute the initial transmission using the expression \( t_0(u, \lambda) = e^{-\beta(\lambda)d_0(u)} \). We also make use of this estimate to define the discrete transmission label set by constraining its values to be within the interval \( [0, \max_\lambda (t_0(\cdot, \lambda))] \).

### 3.6.2.2 Results

In Figure 3.8, we show results for our factor graph method and two alternatives. These are the Bayesian defogging in [135] and the contrast enhancing method in [167]. The figure shows that our albedo estimates are comparable to those yielded by the method in [135]. This contrasts with those delivered by the method in [167], which are often saturated. This is somewhat expected since our graph method is related to the factorial MRF used by the alternative in [135]. Our albedo results also appear more natural with some of the atmospheric effects still contained in the images. This is a result of the dependence of the depth estimate upon the albedo and vice-versa, which avoids over-enhancement due to saturation effects at infinite scene depths, \( i.e. \) in the sky or horizon regions.

Also, note that our depth estimates appear to have smaller unexpected variations between their maximum and minimum estimates. This can be better appreciated in the wheat image at the bottom of the figure. In contrast to the results yielded by our factor graph, the depth for the method in [167] is computed in a post-processing step, so that its depth estimate can become skewed towards the albedo. This effect is clearly visible in the pumpkin image.

### 3.7 Conclusion

In this chapter, we have presented a factor graph method that performs inference by constructing local subtrees of low treewidth from a layered graph comprised of
Figure 3.8: Defogging results. Left-hand column: Real-world images used in our experiments. Middle-left column: Results delivered by the method in [135]. Middle-right column: Albedo and depth obtained using the method in [167]. Right-hand column: Albedo and depth yielded by our factor graph approach.
variable and factor nodes. The method is very general in nature, being applicable
to either lattice-like graphs defined over dependent variables or irregular regions
in many-to-one or one-to-many relationships across layers. Moreover, we have
shown how a Delaunay triangulations can be used to define adjacency relations-
ships between irregular regions within layers. The triangulations are constructed
by a dynamic method that inserts points one-by-one and updating interim trian-
gulations until all points are added. This means that the triangulation method
is much more efficient - converging in order $O(n \log n)$ time for $n$ points - com-
pared to others in literature. Moreover, the method is suitable for the irregular
region-centric graphs encountered in the methods in this chapter. In the next
Chapters 4 and 5, we will consider a different method that is also suitable for the
grid-like graphs that occur there, i.e. the method is applicable to graphs of any
structure in general.

The graphs we have constituted by the triangulation method here have then
been partitioned so as to tackle the global MAP inference process in hand using
a max-product algorithm over the factors in the local sub-graphs. Finally, we
have illustrated the utility of our approach for purposes of image enhancement
and segmentation. We have also compared our approach with other methods
elsewhere in the literature.
Chapter 4

Factor Graphs for Pixelwise Illuminant Estimation

4.1 Introduction

This chapter presents a method to recover a spatially varying illuminant colour estimate from scenes lit by multiple light sources. Starting with the image formation process, we formulate the illuminant recovery problem in a statistically data-driven setting. To do this, we use a factor graph defined across the scale space of the input image. In the graph, we utilise a set of illuminant prototypes computed using a data driven approach. As a result, our method delivers a pixelwise illuminant colour estimate being devoid of libraries or user input. The use of a factor graph also allows for the illuminant estimates to be recovered making use of a maximum a posteriori (MAP) inference process. Moreover, we compute the probability marginals by performing a Delaunay triangulation on our factor graph. We illustrate the utility of our method for pixelwise illuminant colour recovery on widely available datasets and compare against a number of alternatives. We also show sample colour correction results on real-world images.

4.2 Background

Note that, in Chapter 3 we have constructed the Delaunay triangulations by inserting points one-by-one and updating the triangulation dynamically. This algorithm is efficient - achieved in $O(n \log n)$ time for $n$ points - for arbitrary region-centred graphs that are created by the out-region initialisations. In
this chapter, however, we implement the slower\textsuperscript{1} convex hull algorithm described in \cite{183} to cater for grid-like graphs. This is because the algorithm used in Chapter 3 cannot update the triangulation when three points with the same horizontal, or vertical, coordinate are added in succession, a scenario likely to occur in grid-graphs. Furthermore, the inference in Chapter 3 is carried-out on local subtrees built through partitioning the graph. From this chapter onwards, we engage the entire factor graph for the purposes of inference, to maintain adjacency relationships and improve accuracy.

The rest of the chapter is organised as follows. In the next Section 4.3, we survey the literature regarding illuminant estimation. Section 4.4 then commences by presenting the notation and background formalism used throughout the chapter. It also presents the factor graph structure used in this chapter and the inference process used to compute the pixelwise illuminant colour estimate. In Section 4.5, we elaborate upon the practical implementation of our method, its initialisation, and the computational issues regarding the illuminant estimate recovery. In Section 4.6, we present a quantitative and qualitative analysis of our method and compare our results with those yielded by a number of alternatives elsewhere in literature. We also show results on the colour correction of real-world scenes. Finally, we conclude on the developments presented here in section 4.7.

4.3 Review

The apparent colour of an object depends on a number of factors. One of these is the power spectrum of the lights illuminating the scene. As a result, the recovery of the illuminant colour as a means to photometric invariance has found applications in areas such as object recognition \cite{66, 78}, visual surveillance \cite{144}, white balancing \cite{86}, digital media production \cite{119} and visual tracking \cite{190}.

Despite its importance, the recovery and identification of the illuminant colour in a scene has proven to be a difficult task in uncontrolled real world imagery. This is mainly due to the fact that the recovery of the pixelwise illuminant from a single

\textsuperscript{1}The algorithm is known to be $O(n^{(2d-1)/d})$ in time for $n$-points, and a $d$-dimensional hull \cite{183}. Recall that the dynamic algorithm used in Chapter 3 was $O(n \log n)$, which is linear in time. More importantly, $O(n \log n)$ time is a lower bound to finding convex hulls \cite{28}.
image is an under-constrained problem \cite{32, 26}. As a result, existing methods often assume uniform illumination across the scene \cite{33, 32, 46}. Furthermore, additional constraints or assumptions are used to render the problem tractable. For instance, the grey-world method \cite{33} renders the mean reflectance across the scene to be equivalent to the illuminant colour. The grey-edge \cite{175} adopts a similar assumption, whereby the mean reflectance difference computed from the image colour derivatives is also assumed to be identical to uniform illuminant colour. The white patch algorithm \cite{109} recovers the illuminant colour which is correlated to the maximum values of each colour band. The shades of grey method \cite{53} builds upon the grey-world and white patch algorithms and assumes the Minkowski norm of the derivatives of the reflectance of objects in the scene is achromatic.

Some of these approaches \cite{32, 152, 63} also employ Bayesian statistics. This is because statistical inference allows for priors and uncertainty to be used for the illuminant recovery process. The classical Bayesian colour constancy method in \cite{32} models the reflectance distribution as a mixture of Gaussians, where the illuminant is recovered using the posterior computed via a minimum risk rule. Rosenberg \textit{et al.} \cite{152} build upon the approach in \cite{32} and employ a histogram to estimate the reflectance distributions using a clipping function. Gehler \textit{et al.} \cite{63} introduce a parameter to control the effect of the bin clipping function in \cite{152}.

Despite being effective, the assumption of a uniform illuminant colour as applied by the methods above implies that, in practice, these approaches can only tackle scenes lit by a single illuminant \cite{26}. This contrasts with real world images, which often depict scenes lit by multiple lights sources. We illustrate this in Figure 4.1, where we show a scene lit by multiple illuminants. Note that, the original image, on the left-hand panel, has several round spot lamps illuminating the paintings on the wall. In the right-hand panel, we show the image where the effects of the illuminant colours have been removed. Note that the background wall no longer exhibits the “warm” tone induced by the lights.

To handle varying illumination, methods such as the retinex algorithm \cite{109, 60} assume the illumination varies smoothly across the scene. This is based upon the notion that the degree of change in independent colour channels can be employed to determine whether these variations correspond to variations in the
Figure 4.1: Left-hand panel: Example real-world image with multiple illuminants; Right-hand panel: Colour corrected result obtained by removing the effects of the illuminants.

illumination colour, or the reflectance of the objects in the scene. Other methods segment the image into regions of constant illumination or perform colour constancy without recovering the illuminant colour explicitly. For instance, Wang and Samaras [179] detect and estimate the illuminants in the scene by making use of a recursive least squares method to segment the surfaces in the scene into separate light patches. Ebner [46], on the other hand, employs the local average colour to perform colour constancy irrespective of the illuminants used in the scene.

To account for illumination variations in the scene, thresholds on the derivative of the logarithm of colour channels have been employed so as to impose a smoothness constraint on the scene irradiance. In [17], Barnard et al. used smoothness constraints on both the reflectance and illumination gamuts to identify varying illumination. The method in [17] employs the white patch algorithm [109] to account for skewed image brightness. In fact, the use of single-illuminant methods for multiple-illuminant recovery is not unusual. For instance, Gijsenij et al. [68] combined the estimates of existing approaches for uniform illuminant colour recovery to account for local regions in the scene. The varying illuminant is then recovered by clustering and merging these single-illuminant regions across the image, so as to arrive to a per-pixel illuminant estimate. Gu et al. [73], on the other hand, group pixels into regions that jointly maximise the weighted sum of the illuminants in the scene and the likelihood of the associated image reflectance.
Once the pixels have been grouped into regions with a similar illuminant, these can be processed using existing single illuminant colour recovery methods. In a related development, Bleier et al. [26] segment the image into superpixels using albedo and edge information so as to apply single illuminant methods efficiently and effectively. The method of Riess et al. [146] also segments the image into regions of uniform illumination, but uses the inverse-intensity chromaticity (IIC) space [168] for clustering pixels accordingly.

Other multiple illuminant algorithms impose constraints on the number of lights that lit the scene [86], employ specialised hardware [51] or user inputs [29, 30]. For instance, Hsu et al. [86] solve the white balance problem by restricting the scene to two known illuminant colours and estimating the illuminant mixture components using Laplacian matting [112]. Recently, Beigpour et al. [19] have used a conditional random field (CRF) that combines local illuminant interactions with their global spatial distribution. In their approach, image patches are classified as either diffuse or specular, and an illuminant label set is obtained through K-means clustering of the two sets. The illuminant recovery task is then posed as an energy minimisation problem whose aim of computation is to find a dominant pair of illuminant labels that correspond to the optimal maximum a posteriori (MAP) labelling over the CRF.

Here, we present a statistical method for multiple illuminant colour estimation based upon a factor graph [108]. This treatment allows for subgraphs to be constructed using a pyramid-like structure where the strata of each of these is given by the image lattice at different scales. We initialise each of these subgraphs using estimates delivered by existing colour constancy methods and view the pixelwise illuminant as a weighted linear combination of a set of prototype illuminants. This linear combination is such that the weight of an illuminant prototype at a given pixel corresponds to an a posteriori probability which is derived from our factor graph. Moreover, by construction, the pixelwise illumination at each scale of a subgraph is also a weighted linear combination of the illuminants across the subgraphs comprising our factor graph. Thus, the pixelwise illuminant can be viewed as the geometric mean of the illuminants across all subgraphs. In this manner the pixelwise illumination is the result of a statistical inference process across all subgraphs. Note that our method is data driven, whereby the
prototype illuminants are determined statistically from the image itself without the need for additional constraints on the reflectance or illumination across the scene.

4.4 Illuminant Estimation

4.4.1 Background

As mentioned earlier, we start with the image formation process to develop our pixelwise illuminant recovery approach. This allows for an appropriate motivation and better understanding of our factor graph, its subgraphs and strata, as spanned by the image scale space and initial estimates delivered by existing colour constancy methods. Thus, in this section, we will provide some background and notation used throughout the chapter regarding the scale space used for our factor graph, the image irradiance and the manner in which the pixelwise light colour estimates can be expressed as a linear weighted combination of illuminant prototypes.

To commence, recall that our factor graph is divided into subgraphs, each of which corresponds to a particular initialisation. The strata of each subgraph correspond to a different scale across the image. We have done this so as to account for the variations of the illuminant at different resolutions in the image. It is worth noting in passing that this treatment is consistent with the behaviour of the human visual system whereby information about the scene at equally distinct resolutions is used to sense different structural aspects in the image relative to its surroundings, a concept known as lateral inhibition [150]. Also, note that the notion of scale has been used in edge based colour constancy [175], where scale is viewed as the spread of a Gaussian distribution. Furthermore, the method in [175] estimates a uniform colour illuminant using the derivative of the image at a given scale. Here, we obtain different scales by smoothing the image data using the Gaussian kernel

\[ G(u; \sigma) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{u^2}{2\sigma^2}}, \]  

(4.1)

where the variance \( \sigma \) represents the scale.
Figure 4.2 shows the effect of the scale $\sigma$ in the Gaussian kernel in Equation (4.1) upon the illuminant recovered by the method of Gijsenij et al. [68]. The figure shows a wall lit by two illuminants of different colour. Note that, on the right-hand panel, the area covered by illuminants “widens” when the image is blurred. This can be noticed as the area of grey-coloured light spreads out around the lamp towards the reflection on the wall. Likewise, in the bottom-left corner, the orange hue overlaps onto the multi-coloured scene details, which are much more clearly visible at the finer scale in the left-hand panel.

For each scale, consider the spectral radiance $I_{\sigma}(u, \lambda)$ at pixel $u$ and wavelength $\lambda$ of a scene lit by a pixelwise illuminant with power spectrum $L_{\sigma}(u, \lambda)$. With this notation, we can express the scene radiance as follows

$$I_{\sigma}(u, \lambda) = L_{\sigma}(u, \lambda)R_{\sigma}(u, \lambda),$$  \hspace{1cm} (4.2)$$

where $R_{\sigma}(u, \lambda)$ is a function of the mean scattered power, the surface spectral reflectance and the camera spectral sensitivity function [84]. Note that $R_{\sigma}(\cdot)$ above accounts for the proportion of the incident light that is reflected by the
object at a particular scale $\sigma$. This function hence defines the colour of the object at the scale under consideration. Further, the relation above has been widely used in colour constancy [47] and is consistent with the dichromatic model [52].

As mentioned earlier, each of the subgraphs in our factor graph is initialised using a set of prototypes delivered by an existing method. This is done using a set of prototypes $\Upsilon_k$ drawn from the image itself when processed by the $k^{th}$ existing approach under consideration, i.e. white patch [109], shades of gray [53], etc. Making use of these prototypes, we can express the illuminant at a pixel location $u$, and at a subgraph initialised with method $k$ as follows

$$L_{\sigma,k}(u) = \sum_{i \in \Upsilon_k} \omega_{i,\sigma}(u) \ell_i,$$  \hspace{1cm} (4.3)

where $\omega_{i,\sigma}(u)$ is the weight of the $i^{th}$ prototype illuminant $\ell_i \in \Upsilon_k$. Also, from now on and throughout the chapter, for the sake of clarity and without loss of generality we have omitted the wavelength variable $\lambda$ and the subscript $k$ on the weights $\omega_{i,\sigma}(u)$ and the prototype $\ell_i$.

The expression in Equation (4.3) implies that the illuminant at a pixel depends on the contributions of the data-driven illuminant prototypes $\ell_i$ to the power spectrum of $L_{\sigma,k}(u)$. It is worth noting that such a constrain upon the pixelwise illuminant estimates is not overly restrictive. This is because the prototypes in $\Upsilon_k$ are pre-computed from the input image, as opposed to being chosen from a library or user input. This has the advantage that these prototypes are, by definition, drawn from the image itself and, therefore, are expected to be in good accordance with the input data. Here, we have computed these prototypes making use of the mode seeking method in [177].

### 4.4.2 Factor Graphs

With the notation and background above, we now proceed to consider the weight $\omega_{i,\sigma}(u)$ in Equation (4.3) as a probability and lay out the illuminant recovery problem in a manner akin to maximum a posteriori estimation using a factor graph [108].

Recall that a factor graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ is an undirected graphical model comprised of a node set $\mathcal{N}$ and an edge set $\mathcal{E}$. Its node set is subdivided into two
distinguishable subsets. The first of these is given by the variables \( V \) whereas the other one contains the factors \( F \), i.e. \( N = V \cup F \). The edge set connects variables to factors so as to parameterise the conditional probabilities of the cliques that constitute the graph.

As mentioned earlier, our factor graph is comprised by a set of subgraphs with image scales as strata. Figure 4.3 depicts the factor graph arising from the relations between the illuminant variables and the image scale space. In the left-hand panel, we show the interaction between variables initialised by the prototypes arising from different illuminant recovery methods as subgraphs, whereby each of these bears a pyramid-like structure induced by the multiple scales used over the image lattice. Note that there is a one-to-one relationship between the variables across adjacent subgraphs, while the connection from finer to coarser scales is many-to-one. Additionally, the right-hand panel of Figure 4.3 shows a
close up of one of the subgraphs in the left-hand panel. In the figure, the factors are represented by square tokens whereas the variables are given by the circular markers along the image lattice across different scales.

In this chapter, we view the pixelwise illuminant estimates $L_{\sigma,k}(u)$ at each of these subgraphs as variables, i.e. $V = \bigcup_{u,\sigma,k} L_{\sigma,k}(u)$. The factors, on the other hand, can be viewed as the potential functions $\psi(L_c)$ which define the relationship between a subset of connected variables, i.e. over a clique $c$. Note that in coding theory and low-density parity-check codes [122, 117], these potentials are multivariate functions of Fourier transforms. Here, the factorisations are probabilities governed by the image data and the illuminant prototype set $\Upsilon_k$.

Moreover, from a probabilistic standpoint, we can view the interrelated variables of the graphical structure in the right-hand side of Figure 4.3 as conditional distributions within and across scales of each subgraph. In this setting, if $\mathcal{G}_k$ denotes a factor subgraph for the $k^{th}$ colour constancy method employed at initialisation, the complete graph $\mathcal{G}$ can be fully written as the union over the set of subgraphs $\mathcal{K}$ given by

$$\mathcal{G} = \bigcup_{\mathcal{G}_k \in \mathcal{K}} \mathcal{G}_k. \quad (4.4)$$

This is important since the probability of the pixelwise illuminants across the image $\mathbf{L}$ given the graph $\mathcal{G}$ can be viewed as the product of the potentials over the clique set $C$ of the graph. This, in turn, can be expressed in terms of the set of subgraphs $\mathcal{K}$ and the pixelwise illuminant $L_k(u)$ for the $k^{th}$ subgraph as follows

$$P(\mathbf{L}|\mathcal{G}) = \frac{1}{Z} \prod_{c \in C} \psi(\mathbf{L}_c; \bigcup_{\mathcal{G}_k \in \mathcal{K}} \mathcal{G}_k)$$

$$= \frac{1}{Z} \prod_{\mathcal{G}_k \in \mathcal{K}} \prod_{u,v \in \mathcal{G}_k} P(I(u), I(v)|L_k(u), L_k(v)) \prod_{\sigma \in \Gamma} \prod_{\ell_i \in \Upsilon_k} P(L_{\sigma,k}(u)|\ell_i), \quad (4.5)$$

where $\Gamma$ is the set of scales under consideration, $C$ is the clique-set of all factors in the graph, $u \sim v$ means that pixels $u$ and $v$ are neighbouring each other, $\psi(\mathbf{L}_c; \mathcal{G}_k)$ is the potential function over connected variables in the clique $c \in C$ and $Z$ is a partition function.
The nature of Equation (4.5) is such that it effectively conditions the illuminant distribution across the image to the potential functions for the variable nodes over the graph and the prototypes in $\Upsilon_k$. This is because each of the potential functions is defined over a subset of variables, i.e. the pixelwise illuminants $L_c$ for the clique $c \in C$. This has the advantage that it ensures every variable in a clique is connected to at least one other variable through a factor node. At the factor node, the conditional probabilities associated with the corresponding variables are defined by taking the sets of variables, factors and the edges that connect them together across the factor graph $G$.

It is also worth noting that the expression in Equation (4.5) is motivated by Bayes’ rule. This reflects the notion that the pixelwise illuminant recovery problem can be stated in terms of the likelihood and prior distributions with the partition function $Z$ acting as a normalisation constant. Here, we assume that the illuminant priors $P(L_{\sigma,k}(u)|\ell_i)$ for our prototype illuminants across image scales $\sigma \in \Gamma$ and subgraphs $G_k \in K$ are independent from each other. On the other hand, adjacent pixels on each scale relate to factor potentials which, in turn, define the likelihood function for the illuminant estimation process. The likelihood function $P(I(u), I(v)|L_k(u), L_k(v))$ captures the most likely state of the observed image data at neighbouring pixel sites given their respective illuminant estimates. Additionally, the priors $P(L_{\sigma,k}(u)|\ell_i)$ correspond to the likelihood of an illuminant for a given pixel $u$ at scale $\sigma$ to assume the illuminant prototype $\ell_i$ from the prototype set $\Upsilon_k$.

It is worth mentioning that, here, we have omitted the probability $P(L_k(u)|L_{k-1}(u))$ which captures the relationship between subgraphs. Later on, in Section 4.5.1, we will elaborate further on this probability and its role in our inference process. For now, our attention remains on the Bayesian formulation in Equation (4.5). This is as it lends itself, by construction, to a probability factorisation over the graph. This is important since the inference process can now be posed as maximum a posteriori (MAP) inference over the subgraphs. The objective hence becomes, to compute marginal probabilities $P(L_{\sigma,k}(u))$ for the variable set in the
graph. This can be expressed in an optimisation setting as follows
\[
\arg\max_L P(L|G_k) = \arg\max_L \left\{ \prod_{g_k \in \mathcal{K}} \prod_{u,v \in g_k} P(I(u), I(v)|L_k(u), L_k(v)) \prod_{\sigma \in \Gamma} \prod_{\ell_i \in \Upsilon_k} P(L_{\sigma,k}(u)|\ell_i) \right\}
\]
(4.6)

where the partition function $Z$ has been removed from further consideration since it is a constant that does not affect the maximisation above.

### 4.4.3 Priors and Probabilities

The prior conditional probability of the target function in Equation (4.6) makes use of the cosine of the Euclidean angle between the pixelwise illuminant and each of the prototypes in $\Upsilon_k$. This is, effectively, a similarity measure between an illuminant at the pixel $u$ and the $i^{th}$ prototype illuminant $\ell_i$ given by
\[
\cos \theta = \alpha_{u,i} = \frac{\langle L_{\sigma,k}(u), \ell_i \rangle}{\|L_{\sigma,k}(u)\| \|\ell_i\|},
\]
(4.7)

where $\theta$ is the angle between the two vectors, $\langle \cdot, \cdot \rangle$ is the inner product and $\| \cdot \|$ is the vector $L2$ norm.

For the illuminant prior, we ensure the cosine above maps to a probability by setting
\[
P(L_{\sigma}(u)|\ell_i) = \frac{1}{Y} \exp \left( -1 - \alpha_{u,i} \right),
\]
(4.8)

where $Y$ is a normalisation constant given by
\[
Y = \sum_{\ell_i \in \Upsilon_k} \exp \left( -1 - \alpha_{u,i} \right).
\]
(4.9)

For the likelihood in Equation (4.6), we adopt the notion that neighbouring pixels are likely to have similar illuminant colour if their reflectance and shape are close to one another. Recall that the variable $R_{\sigma}(u, \lambda)$ in Equation (4.2) can be expressed in terms of the image irradiance and the illuminant power spectrum.
at pixel $u$.

As a result, the likelihood in Equation (4.6) can be written as follows

$$P(I(u), I(v)|L_k(u), L_k(v)) = \frac{1}{\Omega} K(L_k(u), L_k(v)) \times \exp \left( -\|R(u, \cdot) - R(v, \cdot)\|^2 \right),$$

where we have omitted the variable $\sigma$ as a matter of convenience and $\Omega$ is a normalisation constant given by

$$\Omega = \sum_{u, v \in G_k} K(L_k(u), L_k(v)) \exp \left( -\|R(u, \cdot) - R(v, \cdot)\|^2 \right).$$

In Equation (4.11), $R(u, \cdot)$ is a vector spanning the whole set of wavelengths under consideration at a given scale and $K(L_k(u), L_k(v))$ is a kernel function.

Note that the formulation above is quite general and admits the use of a variety of kernel functions. Here, we explore the use of three different kernels. The first of these is the similarity measure discussed earlier based upon the cosine, i.e.

$$K(L(u), L(v)) = \exp \left( -(1 - \alpha_{u,v}) \right)$$

where $\alpha_{u,v}$ is the cosine of the angle between the illuminants at adjacent pixels defined as

$$\alpha_{u,v} = \frac{\langle L(u), L(v) \rangle}{\|L(u)\| \|L(v)\|}.$$

The other kernel functions used here are the probabilistic variants for two popular M-estimators often used in robust statistics [88]. These robust estimators ensure the pixelwise illuminants are not overly affected by unwanted outliers. These are the Huber estimator [88] given by

$$K(L(u), L(v)) = \begin{cases} \exp \left( -\alpha_{u,v}^2 \right), & \text{if } |\alpha_{u,v}| \leq \tau, \\ \exp \left( -(2\tau |\alpha_{u,v}| - \tau^2) \right), & \text{if } |\alpha_{u,v}| \geq \tau. \end{cases}$$
and the Tukey estimator [174]

\[
K(L(u), L(v)) = \begin{cases} 
\exp \left( -\frac{\tau^2}{6} \left( 1 - \left[ 1 - \left( \frac{\alpha_{u,v}}{\tau} \right)^2 \right]^3 \right) \right), & \text{if } |\alpha_{u,v}| \leq \tau \\
\exp \left( -\tau^2 / 6 \right), & \text{if } |\alpha_{u,v}| \geq \tau
\end{cases}
\] (4.15)

where, for both kernels, \( \tau \) is a cut-off variable.

Note that, in contrast with the non-probabilistic estimators used elsewhere, here we have used an exponential function to bound the kernels in Equations (4.14) and (4.15) above to the range \([0, 1]\). We have done this without loss of generality following the notion that the likelihood in Equation (4.10) should satisfy the rules of probability.

### 4.4.4 Inference Process

With the probabilities above in hand, we proceed to solve the MAP inference problem in Equation (4.6). Note that the aim of computation in each subgraph is the set of marginal probabilities \( P(L_{\sigma,k}(u)) \) for the pixelwise illuminant at the finest scale. In this chapter, we employ the sum/max-product algorithm in [108], this is, effectively, a message passing strategy that entails exchanging information
so as to update the probability distributions encountered throughout the graph.

Recall that the sum/max-product algorithm computes exact probability marginals of variables in a tree [138, 6]. In order to take advantage of this property, we construct our factor graph as a tree. Note that grid-graphs arising from lattices, such as the one used here, are often converted into trees using variable elimination techniques [196, 43]. These methods introduce fill-edges in lattice-like graphs, thus generating cliques with a large number of variables, i.e. with a large tree width. As a result, the computational burden can potentially increase greatly when the clique potential is computed. Moreover, the large tree width effectively draws many variables into a few cliques, changing the topology of the graph under study and potentially changing the inference problem in hand [15].

Consequently, we perform variable elimination using a convex hull algorithm for Delaunay triangulations [183]. We choose Delaunay triangulations motivated by the fact that they place adjacency constraints on the graph - i.e. the longest cycle in the graph is of length three. Thus, we have a limited treewidth\(^2\) which preserves the topology of the graph. This also ensures computational efficiency during message passing operations across the graph.

In Figure 4.4, we show the factor graph corresponding to the MAP problem in Equation (4.6). In the left-hand panel of the figure, we illustrate the interaction between variables across different scales after a Delaunay triangulation is computed. Note that this structure is consistent with the pyramid-like subgraphs in Figure 4.3. In the right-hand panel of Figure 4.4 we show the detail of the left-hand panel, where we have included the messages relayed between variables and factors. In the figure, we have adopted the notation used throughout the rest of the chapter. The variables \(x\) and \(y\) denote the illuminant variables at different scales, respectively. The factor symbols are \(f\) and \(g\), and \(\mu\) indicates a message flowing in the direction of the sub-indexed arrow.

More explicitly, note the factor denoted by \(f\) “passes on” the probability information \(\mu_{f \rightarrow x}\) to the variable \(x\) according to the expression

\[
\mu_{f \rightarrow x} = \max_{\mathcal{X}\setminus x} F(\mathcal{X}) \prod_{\substack{y \in \mathcal{N}_f \\ y \neq x}} \mu_{y \rightarrow f},
\]

\(4.16\)

\(^2\)The treewidth of a graph is the number of variables in its largest clique.
where $\mathcal{X} = \mathcal{N}_f$ denotes the set of all variable nodes adjacent to the factor $f$ and $F(\mathcal{X})$ is the probability distribution at the factor. Our message passing scheme entails computing probability distributions and marginalising across subgraphs. Consequently, the definition of the distribution $F(\mathcal{X})$ in Equation (4.16) at the factors is determined by the nature of the node, i.e. either inter-scale or within-scale. We illustrate these messages in the two left-most panels of Figure 4.5.

The messages from the variable node $x$ to a factor node $f$ is deemed to be

$$
\mu_{x \rightarrow f} = \prod_{g \in \mathcal{N}_x \setminus f} \mu_{g \rightarrow x},
$$

(4.17)

where $\mathcal{N}_x$ is the set of factor nodes neighbouring the variable node $x$.

In the right-most panel of Figure 4.5 we show the diagram for the variable-to-factor message passing in Equation (4.17). Note that, so far, we have focused our attention on the intra-scale and within-scale factor as defined in Equations (4.8) and (4.10). The inter-subgraph messages are derived from Equation (4.10), where neighbouring pixels are between two subgraphs rather than across scales. The explicit formulation is as follows,

$$
P(I_{k-1}(u), I_k(u)|L_{k-1}(u), L_k(u))
= \frac{1}{\Omega} K(L_{k-1}(u), L_k(u)) \times \exp \left( -\|R_{k-1}(u) - R_k(u)\|^2 \right),
$$

(4.18)
where, as before, we have omitted the variable $\sigma$ and $\Omega$ is a normalisation constant, $K(L_{k-1}(u), L_k(u))$ is a kernel function and $k$ is the index for the subgraph $G_k$.

The distribution above can be substituted for $F(\mathcal{X})$ in Equation (4.16) for purposes of message passing. Notice that the message of a given variable can also be computed using a sum-rule to marginalise posteriors in Equation (4.16) over the variables in $\mathcal{X}$ and by excluding the variable $x$ under consideration. After the messages have been sent in both directions on all edges in the graph, each factor is expected to contain the joint posterior probability distributions of the variables adjacent to the factor under consideration. As a result, in order to compute the marginal distribution for any one variable in the clique, we marginalise with respect to the adjacent variables making use of the joint distribution. As a result, the estimate $L_{\sigma,k}(u)$ at a pixel $u$ at the finest scale $\sigma$ for a subgraph $G_k$ can be viewed as the expected value of the illuminant with respect to the prototype set $\Upsilon_k$.

Making use of Bayes’ rule and the notation in Section 4.4.4, we can write

$$P(\ell_i | L_{\sigma,k}(u)) = \frac{P(L_{\sigma,k}(u) | \ell_i) P(\ell_i)}{P(L_{\sigma,k}(u))}$$

(4.19)

The probability $P(\ell_i | L_{\sigma,k}(u))$ in the equation above is, in practice, the posterior probability of a pixel being lit by the illuminant $\ell_i \in \Upsilon_k$. This, in turn, is equivalent to the distribution $P(L | G_k)$ in Equation (4.6).

Analogously, the most likely illuminant estimate $L_{\sigma,k}(u)$ at a pixel $u$ at scale $\sigma$ for the subgraph $G_k$ is computed using the illuminant’s weights in Equation (4.3). This is given by

$$L_{\sigma,k}(u) = \frac{\sum_{\ell_i \in \Upsilon_k} P(\ell_i | L_{\sigma,k}(u)) \ell_i}{\sum_{\ell_i \in \Upsilon_k} P(\ell_i | L_{\sigma,k}(u))}$$

(4.20)

Here, we employ the finest scale so as to obtain the final illuminant estimate. As such, the final illuminant estimate is obtained by the geometric mean of the finest scale illuminants, i.e.

$$L(u) = \left[ \prod_{G_k \in \mathcal{K}} L_{\sigma^*,k}(u) \right]^{\frac{1}{|\mathcal{K}| - 1}}$$

(4.21)
Algorithm 4 - Pixelwise Illuminant Recovery

Require: \( I(u) \): image data; \( \sigma \in \Gamma \): set of Gaussian scales; \( b \): bandwidth parameter for the quick-shift method \( \text{[177]} \); \( \tau \): kernel cut-off variable

Ensure: Illuminant estimate \( L(u) \)

1: for all \( k \in K \) do
2: \hspace{1em} for all \( \sigma \) do
3: \hspace{2em} Initialise illuminant estimate according to Algorithm 5 at scale \( \sigma \)
4: \hspace{1em} end for
5: Construct graph \( G_k \) corresponding to the \( k \text{th} \) colour constancy method under consideration
6: Triangulate graph \( G_k \) using the convex hull technique in \( \text{[183]} \)
7: for all Edges \( e \in E \) connecting variables in \( V \) and factors in \( F \) do
8: \hspace{1em} Perform message passing using Equations (4.17) and (4.16)
9: end for
10: Find \( L_{\sigma,k}(u) \) using the probabilities \( P(\ell_i|L_{\sigma,k}(u)) \) at the finest scale as shown in Equation (4.20).
11: end for
12: return \( L(u) \) given by the geometric mean in Equation (4.21)

where \( \sigma^* \) is the finest scale for the \( k \text{th} \) initialisation method under consideration.

4.5 Implementation

In this section, we elaborate further on the implementation of our method. In Algorithm 4 we show the step sequence of our method.

Note that our method, as described in Section 4.4.2, requires a set of connected subgraphs each with strata representing multiple scales. For each of these subgraphs, a prototype set \( \Upsilon_k \) and initial illuminant variables \( L_{\sigma,k}(u) \) must be computed at initialisation. This is done in Line 3 of Algorithm 4 where we have introduced a call to Algorithm 5.

4.5.1 Initialisation

Algorithm 5 summarises how these prototypes and the initial illuminant estimates, denoted \( L_{\sigma,k}^{(0)} \) are computed using the Gaussian kernel in Equation (4.1).

Here, we commence by sub-dividing the image at each given scale into evenly sized patches. Next, we assume the illuminant to be uniform in each patch.
Algorithm 5 - Compute illuminant prototypes $\ell_i$, $\forall k$ and initial illuminant estimates

**Require:** $I(u)$: input image; $k \in \mathcal{K}$: illuminant yielded by the $\mathcal{K}$ methods under consideration; $\sigma \in \Gamma$: set of Gaussian scales; $b$: bandwidth parameter for the method in \cite{177}.

**Ensure:** Illuminant prototypes $\forall k, \ell_i \in \mathcal{Y}_k$

1:   for all $k \in \mathcal{K}$ do
2:       for all $\sigma \in \Gamma$ do
3:           Filter image using the Gaussian kernel in Equation (4.1) at scale $\sigma$, $\mathcal{I}_\sigma \leftarrow I$
4:           Partition $\mathcal{I}_\sigma$ into patches, $\bigcup_m \mathcal{I}_{\sigma,m} \leftarrow \mathcal{I}_\sigma$
5:           for all $\mathcal{I}_{\sigma,m} \in \mathcal{I}_\sigma$ do
6:               Compute uniform illuminant using the $k^{th}$ algorithm in $\mathcal{K}$
7:           end for
8:       $L_{\sigma,k}^{(0)} \leftarrow \mathcal{I}_\sigma$
9:   end for
10:  Apply the method in \cite{177} to all $\mathcal{I}_\sigma$ to find $\ell_i \in \mathcal{Y}_k$
11:  end for
12:  return $\bigcup_k \{\mathcal{Y}_k\}, \bigcup_{\sigma,k} \{L_{\sigma,k}^{(0)}\}$

and use an existing colour constancy method to recover the illuminant for the patch under consideration. The choice of method is the same for all patches and all scales in each subgraph, but the methods differ from one subgraph to the other. Here, we have used 4 subgraphs so as to construct our graph, each of these corresponding to the estimates delivered by the grey world \cite{33}, white patch \cite{109} and the 1st and 2nd order grey edge methods \cite{175}. Once the initial estimates $L_{\sigma,k}(u)$ for all scales $\sigma$ of a subgraph $G_k$ are in hand, we proceed to compute the prototypes $\ell_i$ using the kernel mode seeking algorithm \cite{177}.

Thus, a complete graph will have four subgraphs, each of these comprised by strata corresponding to a scale $\sigma$. Note that the structure of our graph is quite general and can employ other scaling schemes or colour constancy methods. Moreover, it is possible to implement more complex methods for the recovery of the prototype set for each graph as an alternative to the quick shift in \cite{177}. Indeed, the prototypes may be recovered using the contour based region grouping method of Arbelaez et al. \cite{10} or that in \cite{178}. Moreover, pixelwise illuminant initial estimates can be obtained by considering small neighbourhoods about each pixel. Here, however, we have opted for a uniform grid-based approach so as to
Factor Graphs for Pixelwise Illuminant Estimation

<table>
<thead>
<tr>
<th></th>
<th>Gijsenij et al. [68] dataset</th>
<th>1st order GE</th>
<th>2nd order GE</th>
<th>GW</th>
<th>WP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our method</td>
<td>3.10 (2.68)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gu et al. [13]</td>
<td>3.26 (3.25)</td>
<td>3.25 (3.27)</td>
<td>3.89 (3.46)</td>
<td>3.20 (2.97)</td>
<td></td>
</tr>
<tr>
<td>Gijsenij et al. [68]</td>
<td>7.17 (6.52)</td>
<td>6.81 (5.98)</td>
<td>7.06 (6.33)</td>
<td>6.26 (6.27)</td>
<td></td>
</tr>
<tr>
<td>MIRF</td>
<td>7.10 (4.70)</td>
<td>7.20 (5.00)</td>
<td>10.00 (10.10)</td>
<td>7.70 (6.40)</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1: Mean and median per-pixel Euclidean angle error (in degrees) for our method and the alternatives when applied to the natural images in Gijsenij et al. [68] dataset. The median is shown in parenthesis and the absolute best performance is written in bold font.

keep the computational burden low.

4.5.2 Illuminant Recovery

Once the factor graph has been constructed and initialised, we proceed to compute a Delaunay triangulation across the three-level pyramid comprising each of the subgraphs $G_k$. With the triangulated graph in hand, we then apply the message passing algorithm in Section 4.4.4 relying on the probability distributions in Equations (4.8) and (4.10). This is so as to solve the MAP problem in Equation (4.6) and recover the marginals $P(L_{\sigma,k})$ at the finest scale of each subgraph. These marginals are then used as an additional “prior” in Equation (4.18).

Once the inference process has been performed on all subgraphs, the posterior probabilities $P(\ell_i|L_{\sigma,k}(u))$ from each subgraph can be used to compute the pixelwise illuminant using Equation (4.19) in Section 4.4, where $\frac{P(\ell_i)}{P(u)}$ is set to be a constant in Equation (4.20). This follows the intuition that every pixel and prototype are equally likely. With the posteriors in hand, the pixelwise illuminant can be computed in a straightforward manner using the geometric mean in Equation (4.21).

4.6 Experiments

We now turn our attention to the utility of our method for pixelwise illuminant estimation of a single image and provide comparison to other methods elsewhere in literature. To this end, we have made use of four image datasets acquired under multiple light sources.
### §4.6 Experiments

#### Table 4.2: Mean and median per-pixel Euclidean angle error (in degrees) for our method and the alternatives when applied to the natural images in the Beigpour et al. [19] dataset. The median is shown in parenthesis and the absolute best performance is written in bold font.

<table>
<thead>
<tr>
<th></th>
<th>1st order GE</th>
<th>2nd order GE</th>
<th>GW</th>
<th>WP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Our method</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.48 (2.96)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gu et al. [73]</td>
<td>3.54 (3.16)</td>
<td>3.54 (3.16)</td>
<td>4.43 (3.86)</td>
<td>3.91 (3.33)</td>
</tr>
<tr>
<td>Gijsenij et al. [68]</td>
<td>4.80 (4.20)</td>
<td>5.90 (5.70)</td>
<td>6.40 (5.90)</td>
<td>5.10 (4.20)</td>
</tr>
<tr>
<td>MIRF</td>
<td><strong>3.03 (2.81)</strong></td>
<td>3.19 (3.09)</td>
<td>3.25 (2.95)</td>
<td>3.28 (2.93)</td>
</tr>
</tbody>
</table>

#### Table 4.3: Mean and median per-pixel Euclidean angle error (in degrees) for our method and the alternatives when applied to the Bleier et al. [26] dataset. The median is shown in parenthesis and the absolute best performance is written in bold font.

<table>
<thead>
<tr>
<th></th>
<th>1st order GE</th>
<th>2nd order GE</th>
<th>GW</th>
<th>WP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Our method</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.95 (2.90)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gu et al. [73]</td>
<td>3.32 (3.39)</td>
<td>3.23 (3.32)</td>
<td><strong>1.16 (1.18)</strong></td>
<td>1.86 (1.77)</td>
</tr>
<tr>
<td>Gijsenij et al. [68]</td>
<td>14.52 (14.89)</td>
<td>13.82 (13.89)</td>
<td>4.93 (4.71)</td>
<td>6.04 (6.12)</td>
</tr>
<tr>
<td>MIRF</td>
<td>7.60 (6.93)</td>
<td>7.37 (6.79)</td>
<td>6.53 (6.20)</td>
<td>7.94 (7.67)</td>
</tr>
</tbody>
</table>

#### Table 4.4: Mean and median per-pixel Euclidean angle error (in degrees) for our method and the alternatives when applied to our dataset. The median is shown in parenthesis and the absolute best performance is written in bold font.

<table>
<thead>
<tr>
<th></th>
<th>1st order GE</th>
<th>2nd order GE</th>
<th>GW</th>
<th>WP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Our method</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>4.17 (3.34)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gu et al. [73]</td>
<td>7.54 (6.65)</td>
<td>7.39 (6.56)</td>
<td>7.62 (6.69)</td>
<td>8.70 (7.79)</td>
</tr>
<tr>
<td>Gijsenij et al. [68]</td>
<td>10.60 (8.77)</td>
<td>10.27 (8.61)</td>
<td>7.95 (3.40)</td>
<td>5.79 (3.74)</td>
</tr>
<tr>
<td>MIRF</td>
<td>10.01 (8.93)</td>
<td>10.23 (8.95)</td>
<td>8.90 (7.86)</td>
<td>8.79 (7.69)</td>
</tr>
</tbody>
</table>

The first of these is the image dataset of natural outdoor scenes captured by Gijsenij et al. [68]. The ground truth colour for this dataset was acquired by positioning several grey surfaces on the scene and manually annotating the corresponding grey-world estimate for each illuminant. The second dataset used here is the multiple-illuminant multi-object (MIMO) dataset presented in [19]. This comprises 78 images of scenes lit with multiple illuminants and taken in both controlled laboratory, and real-world outdoor environments. In contrast

---

3The dataset is accessible at http://colorconstancy.com/?p=659
to the first dataset, the ground truth for this imagery was obtained per pixel, taking into account the contribution of the illuminants in the scene. The third dataset is the multi-illuminant image dataset of Bleier et al. [26]. It consists of four scenes acquired in two-illuminant lighting setups, where each lamp can bear several colour filters. The ground truth information for the dataset in [26] was acquired by painting the scene grey so as to suppress the surface albedo. Finally, we have also used a dataset of 135 images of five scenes acquired in house. Each scene is illuminated by two lights at a time, each of these configured in 27 different combinations of colour attained using Lee filters. Similarly to [26], we have obtained the illuminant colour ground truth by painting each scene with grey paint. In Figure 4.6 we show one sample image from each of the datasets used in our experiments.

For our dataset, we used a kernel bandwidth of 0.15 for the method in [177] to generate a small set of illuminant prototypes. For the datasets in [19] and that of Gijsenij et al. [68], we have set the bandwidths to 0.1 and 0.05, respectively. For the dataset of Bleier et al. [26], we have used a bandwidth of 0.2. Note that the robust kernels in Equations (4.14) and (4.15) require the cut-off variable $\tau$ to be set. For our dataset we have used $\tau = 0.75$, while for the the Beigpour et al. [19] and Gijsenij et al. [68] datasets, we employed $\tau = 0.8$ and $\tau = 0.9$ respectively. For the scenery of the Bleier et al. [26] we have set the value of $\tau$ to 0.9.

Here, we have compared our results against those yielded by a number of alternative methods. These alternatives are, the method of Gijsenij et al. [68], the multi-illuminant recovery method of Gu et al. [73] and the Multi-Illuminant Random Field (MIRF) algorithm presented in [19]. Note that these alternatives employ a variety of existing colour constancy methods as an integral part of the illuminant recovery process. Thus, the result yielded by these alternatives is dependent upon the colour constancy method used. This contrasts with our approach, which utilises all of them to initiate the subgraphs comprising our factor graph simultaneously and, hence, delivers a single result which accounts for the prototypes computed from all these methods. In this chapter, to initialise our factor graph and perform experiments with the alternatives, we use the Grey-World (GW) [33], the first and second order Grey-Edge (1$^{st}$ GE and 2$^{nd}$ GE) [175] and the White Patch (WP) [109] algorithms.
§4.6 Experiments

Figure 4.6: Top row: Example images from the Gijsenij et al. [68] (left-hand panel) and the Beigpour et al. [19] dataset (right-hand panel); Bottom row: Example image from the Bleier et al. [26] (left-hand panel) and our dataset (right-hand panel).

For both our method and that of Gijsenij et al. [68], we have used a grid over the image lattice with a patch size of $10 \times 10$ pixels. Also, for the method in [68], we have set the number of the illuminants in the scene through back-propagation, as reported by the authors. In order to determine the optimal parameters of the MIRF algorithm of [19], we have applied two-fold cross-validation by randomly splitting the images of each dataset into two equally sized subsets for training and testing. Note that, while this method generates an arbitrary size of possible illuminants, it only estimates optimally two illuminants. This contrasts with our pixelwise estimation approach, which neither requires the number of illuminants to be known, nor restricts the illuminants to be the dominant pair of illuminants.

In Tables 4.1, 4.2, 4.3 and 4.4 we present quantitative results for our method and the alternatives for the datasets under study. Note that our method achieves a single result with respect to the Grey-World (GW) [33], the first and second order Grey-Edge (1$^{\text{st}}$ GE and 2$^{\text{nd}}$ GE) [175] and the White Patch (WP) [109] algorithms. In the tables, we show the per-pixel mean and median Euclidean angle, in degrees, between the illuminant chroma delivered by our method and
Table 4.5: Mean Euclidean angle error (in degrees) when a number of different kernels are used in our method as applied to the datasets under study. Absolute best performance per dataset is shown in bold font.

<table>
<thead>
<tr>
<th></th>
<th>Gijsenij et al.</th>
<th>MIMO</th>
<th>Bleier et al.</th>
<th>Our dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cos. dist. (Eq. (4.12))</td>
<td>3.4284</td>
<td>3.4922</td>
<td>3.9977</td>
<td>5.7522</td>
</tr>
<tr>
<td>Huber (Eq. (4.14))</td>
<td>3.3580</td>
<td>3.4807</td>
<td>2.9489</td>
<td>4.1722</td>
</tr>
<tr>
<td>Tukey (Eq. (4.15))</td>
<td>3.1031</td>
<td>3.4807</td>
<td>3.8867</td>
<td>4.1722</td>
</tr>
<tr>
<td>Uniform (Eq. (4.22))</td>
<td>3.4284</td>
<td>3.4922</td>
<td>3.9977</td>
<td>5.6652</td>
</tr>
</tbody>
</table>

Note that our method outperforms all the alternatives when applied to both the Gijsenij et al. [68] and our dataset. On the Bleier et al. [26] and MIMO datasets, our algorithm performance is comparable to the best results yielded by the alternatives. This is expected since our method uses all the colour constancy methods under consideration for purposes of initialisation, thereby delivering a unified result. Also, note that, in most cases, Gijsenij et al’s method [68] does not perform as well as the others. This may be due to the assumption of a single constant illuminant at every sampled patch. Its white patch variant is, however, quite competitive over all the datasets. The MIRF method of [19] performs best on the MIMO dataset, with its results on other datasets often correlated with the method of Gijsenij et al. [68]. This is because the method in [19] obtains two dominant illuminants per scene, with each pixel being lit by a single light. The method in [73], despite being quite competitive, appears to show the largest variations with respect to the colour constancy method used.

Recall that our method employs the kernel $K(L(u), L(v))$ in equation (4.10) so as to compute the pixelwise illuminant. In Table 4.5, we show the mean Euclidean angle for the kernels described earlier in Section 4.4.3. For purposes of further comparison, we have also included the results yielded by a uniform kernel given by

$$K(L(u), L(v)) = \frac{1}{\beta},$$

(4.22)
where $\beta$ is the number of colour constancy methods used to initialise our subgraphs, i.e. 4.

From Table 4.5, it is evident that the robust estimators provide a margin of improvement. This is expected, since pixels that deviate too far from the available illuminants are penalised accordingly. This is consistent with the bar plots in Figure 4.7, where we show the performance as a function of the datasets under study. From the figure, note that the Tukey estimator in Equation (4.15) consistently outperforms the Huber estimator in Equation (4.14). The only exception of this behaviour is the Bleier et al. [26] dataset. Further, the uniform kernel in Equation (4.22) delivers comparable results to those yielded by the cosine in Equation (4.12). This can be attributed to the prior term in Equation (4.10) favouring the contribution of the image data to the inference process.

Finally, in Figure 4.8 we demonstrate the efficacy of our method for colour correction on natural scenes. To this end, we present both the illuminant maps yielded by our method and the colour corrected images computed using the recov-
Figure 4.8: Top row: Real-world input image; Middle row: Illuminant map recovered by applying our method to the images on the top row; Bottom row: Colour corrected images computed using the colour maps on the middle row.

In the top row, we show four images used at input. These correspond to scenes that exhibit multiple illuminants that evidently “skew” the colour. For instance, in the images on the far-left and far-right columns, the “warm” effect of the lighting induced by the tungsten illuminating the front of either towers and the skylight in the background have both been successfully removed. Likewise, observe that the sunlight reflections induced by the water on the boats in the middle-left image, and the underwater scattering in the middle-right image, have been colour corrected using the illuminant map in the corresponding columns of the middle rows. Note that, in the first two left-hand columns of Figure 4.8, the method does not deliver an evenly distributed illuminant colour across the image. In general, the illuminant maps are in good accordance with the per pixel changes expected from the top row imagery. In the right-hand column, our illuminant map suffers from a colour cast induced by the orange-brown sand whose chroma is almost identical to that of the illuminant. We also observe that, in the middle-left column, our illuminant estimation breaks down in the top-right and bottom-right corners. This, however, is a result from the grey-edge initialisation, which, as expected, yields spurious results when no edge information is available in a given patch.
4.7 Conclusion

In this chapter, we have presented an approach to recover pixelwise illuminant colour for scenes with multiple illuminants which does not require precomputed libraries or user input. To do this, we have departed from the image formation process and formulated the illuminant recovery problem in a statistically data-driven setting based upon a factor graph. Our factor graph consists of a set of subgraphs, each of which employs the scale space of the input image and an existing colour constancy algorithm to perform statistical inference based upon a set of illuminant prototypes. This inference process is obtained by making use of a convex hull algorithm for Delaunay triangulation on the graph. This treatment not only improves the inference, but also preserves the graph topology. In the following Chapter 5, we will utilise the stability and efficacy of these Delaunay triangulations on grid-structured graphs to present a tree-reparameterisation framework. This will also be an inference-improving mechanism, which is demonstrated through an evidence combining setting that includes the segmentation and defogging techniques outlined in Chapter 3.

Once triangulations have supported the attaining of outputs, we have performed a quantitative analysis of our method on four multiple illuminant datasets and compared against a number of alternatives. We have also evaluated the effect of varying the kernel function used by our method and demonstrated the effectiveness of our approach to perform colour correction of real-world images.
5.1 Introduction

In this chapter we introduce an evidence combining approach based upon factor graphs. The method presented here is quite general in nature and exploits the capability of factor graphs to combine results from multiple algorithms which correspond to different generative models or graphical structures. We do this by using layers across the factor graph to represent each of the methods under consideration. For purposes of inference, we represent the graph as a simplicial complex using a convex hull algorithm. This allows us to convert this simplicial complex into a simplicial spanning tree, which corresponds to the reparameterisation of the junction tree of the factor graph. Exact inference can then be performed using the sum/max-product algorithm on the simplicial spanning tree. Furthermore, we employ a Procrustean transformation so as to avoid degenerate cases in the inference process. We illustrate how the method can be used for evidence combining in image enhancement and segmentation tasks.

5.2 Background

In this thesis, we have made use of Delaunay triangulations to enforce adjacency constraints. Chapter \[3\] creates triangulations by ordering the variable nodes or points, and sequentially introducing them to an interim triangulation one by one. In this way the Delaunay triangulation is updated each time a new node
is added, and the procedure is repeated until all nodes have been added. In contrast, Chapter 4 uses a convex hull algorithm which is based on the dual of Delaunay triangulations, the Voronoi diagram. In this setting, any two nodes are considered to circumscribe a hypersphere. We make use of this convex hull framework for the Delaunay triangulations in this chapter since the method is more stable for the grid-structured graphs common in computer vision tasks.

Recall, also, that in Chapter 4 we have aggregated illuminant outputs obtained from different methods using a geometric mean. Note that the method computes the ensembled pixelwise illuminant estimate once the posteriors from all methods have been in computed independently. In this chapter, we present a more elaborate method to combine the evidence from different methods during inference. Furthermore, we ensure that where evidence appears to be degenerate, it is made to relate to the method with the most entropy via a Procrustes transformation. Thus, inference in the methods, which correspond to the factor graph layers, impacts each other through prototype matching across layers. The evidence-combined estimate will also be evaluated using the geometric mean once the algorithm terminates.

The chapter is structured as follows. In Section 5.3, we provide the background on factor graph inference and existing machine learning approaches to motivate combining probabilities or results into a single output. Section 5.4 then introduces the tree parameterisations of factor graphs as used throughout this chapter, and presents the simplicial spanning tree and message passing procedure used here to compute marginal probabilities. We also discuss, in Section 5.4.5, how to avoid degenerate cases where the methods being combined yield beliefs that require the matching of marginals between them. In Section 5.6, we illustrate the utility of the method in two sample computer vision problems, i.e. defogging and image segmentation. We conclude on the developments presented here in Section 5.7.

5.3 Review

Many classical problems in computer vision, pattern recognition and data analytics, such as image segmentation, denoising, dehazing, motion analysis and depth
recovery can be tackled in a number of ways. Methods such as Markov Random Fields \cite{70,135} and belief propagation \cite{50} have been used to cast these problems in a graphical model setting. Other approaches, such as convex optimisation \cite{69} and mean shift \cite{40}, have also proven to be quite popular in the community.

Note that these methods often involve posing the problem in hand as a labelling one aimed at assigning each token under consideration (pixels, patches, features, etc.) to one of the classes so as to optimise a target function. This is based upon an optimality criterion which takes into account the affinity of these tokens and the consistency of the class membership with respect to their neighbouring elements. Moreover, a number of inference problems in computer vision, pattern recognition and machine learning are intrinsically related to graphical representations. These methods often simplify the inference process by factorising the graphical model so as to separate it into hierarchies \cite{100}. This, in effect, separates the inference process into subgraphs so it can be performed using statistical pattern recognition techniques \cite{182,13,2}.

Here, we note that factor graphs \cite{108,117} allow for an efficient computation of the sum-product rule of probabilities and, hence, provide a means for the structural factorisation of a function over several variables. This is an important observation since they can be used to combine similar beliefs as delivered by existing approaches, \textit{i.e.} outputs, into a single output by exploiting their ability to compute sum-products for probabilities which may originate from different graphical structures or generative models. Furthermore, the factor graph models used here are Markovian in nature and have been shown to be, in theory, exactly solvable using clique trees known as junction trees \cite{111,8}. Besides the traditional sum-product rule for junction trees \cite{158}, inference can also be computed using a sum-product update sequence that involves division operations \cite{111,91}. The message-passing-with-division approach involves a reparameterisation of the clique tree as a local representation of joint probabilities over the clique making use of the joint probability at their intersection. This is widely known as the HUGIN algorithm \cite{8}, which provides a computationally efficient way for clique potentials to be updated in a manner such that their intersection, \textit{i.e.} the separator set, always holds up-to-date evidence.

In this chapter, we aim at combining the outputs delivered by multiple meth-
ods so as to build upon their strengths in order to obtain better results. This is a common rationale in the community, where evidence combination has been carried-out using a wide variety of approaches such as boosting [56], ensemble learning [75], Dempster-Shafer theory [4] and decision forests [41]. In the case of classification [56, 75], evidence combination involves incorporating a number of classifiers by assigning weights to predictions from the individual methods in order to determine the new prediction.

Here, we view the outputs arising from similar methods in the community as variables which can be combined accordingly, making use of the probability marginals over the graph. The idea is, hence, to combine the results of multiple algorithms so as to render the inference process in hand more robust to noise corruption, outliers, etc. The method presented here is quite general in nature and can be used to combine the outputs of methods that, despite aimed at the same task, are often used independently.

Thus, we use factor graphs transformed into clique trees and profit from their capacity to combine beliefs arising from different generative models which may not share an equivalence relationship between them. It is worth noting in passing that despite factor graphs having been widely used in error correcting coding [117], their application in other domains is very much limited to signal processing [118, 165], wireless networks [123, 186], segmentation [159, 195], image feature fusion [97] and tracking [2].

5.4 Factor Graphs

Our choice of factor graphs stems from the fact that these are a generalisation of probabilistic graphical models, which allow us to formulate the problem in an evidence combining setting. To do this, we note that, in general, graphical models often operate upon observables (pixels, textons, features, etc.) so as to recover hidden variables (labels, values, states, etc.).

This treatment allows for the use of the max-product algorithm in factor graphs [108] for purposes of inference. Factor graphs generalise both Markov random fields and directed graphs such as Bayesian networks. They also model variable relationships that are not necessarily probabilistic, a characteristic which
has made them useful in coding theory, error correcting codes [122, 117] and signal processing [118, 165].

The left-hand panel of Figure 5.1 shows a factor graph arising from a 2D lattice. These are typical of computer vision problems. Exact inference for such a graph can be determined by creating a chordal graph through a variable elimination sequence and constructing a junction tree. The chordal graph and junction tree resulting from these procedures for the grid-graph are displayed in the middle and right-hand panel of the Figure 5.1. Mathematically, the factor graph shows a factorisation of the joint probability distribution \( P(x) \) according to the Gibbs measure, i.e.

\[
P(x) = \frac{1}{Z} \prod_{c \in C} \psi(x_c), \tag{5.1}
\]

where \( \psi(x_c) \) is a potential function defined over subsets of variables \( x_c \) in the clique \( c \) and \( Z \) is a normalisation constant.

When the graph is converted into a chordal one, the probability in Equation (5.1) can be expressed as follows

\[
P(x) = \frac{\prod_{c \in C} P(x_c)}{\prod_{s \in S} P(x_s)}, \tag{5.2}
\]
where $P(x_c)$ is the joint probability of the clique with the subset of variables $x_c \in x$ and $P(x_s)$ is the joint probability of the variables in the separator set $x_s = x_{c_i} \cap x_{c_j}$, where cliques $c_i$ and $c_j$ are adjacent, i.e. connected by an edge. As such, the number of separator sets is equal to the number of edges in the junction tree. In Appendix C, we show how the Gibbs distribution in Equation (5.1) can be reparameterised so as to arrive at the expression in Equation (5.2). Note that Koller et al. [103] also provide a proof for Equation (5.2) which is based on message passing. We have included the proof in the appendix since it does not depend on the message passing operation over the graph and, hence, it shows that the factorisation above is independent of the traversal operation across the graph. Later on, we will present a method based on message passing so as to perform inference on the graph.

For the junction tree in the right-hand panel of Figure 5.1, the joint probability can be computed making use of the factorised quotient (5.2). This is important since it opens-up the possibility of approximating the distribution $P(x)$ by obtaining a tree via variable elimination across the factor graph.

5.4.1 Simplicial Complexes

If we view the graphical model in the left-most panel of Figure 5.2 as a set of triangles organised in a sequence as obtained from a convex hull or Delaunay triangulation, we can obtain the graph in the middle of Figure 5.2. Note that this corresponds to the reparameterisation of the Gibbs distribution in Equation (5.1) in terms of 2-simplexes, i.e. triangles, that constitute a simplicial complex $\Delta$ with

\[
P(x) = \frac{P(x_1, x_2, x_4)P(x_2, x_4, x_5, x_6)P(x_2, x_3, x_6)}{P(x_2, x_4)P(x_2, x_6)P(x_4, x_5, x_6)} \times \frac{P(x_4, x_5, x_6, x_8)P(x_4, x_7, x_8)P(x_6, x_8, x_9)}{P(x_4, x_8)P(x_6, x_8)} \times P(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9).
\]

\[\text{(5.3)}\]
a boundary $\partial \Delta$ such that

$$P(x) = \frac{P_{\Delta}(x)}{\prod_{(u,v)} P_{\alpha \setminus \partial \Delta}(x(u), x(v))},$$

(5.4)

where $\alpha$ is a 1-simplex, i.e. edge, of the complex $\Delta$, which we call a face. In Equation (5.4), $\alpha \setminus \partial \Delta$ denotes all edges in the interior of the graph.

A simplicial complex $\Delta = (V, K)$, is an object of combinatorial topology made up of variable nodes $V$ and a set $K = \bigcup \kappa$ of subsets $\kappa$. Each element $\kappa$ of the set $K$ consists of $p+1$ connected nodes $[x_0, x_1, ..., x_p]$, and is known as a $p$-simplex. Here, $p = 2$, in which case the $p$-simplices are triangles of connected variables and the faces correspond to edges. Consequently, the joint probability distribution of the simplicial complex $P_{\Delta}(x)$ in Equation (5.4) denotes the product of all joint distributions of simplexes of the triangulation$^2$, i.e.

$$P_{\Delta}(x) = \prod_{\kappa \in K} P(x_\kappa),$$

(5.5)

$^2$Note that here we focus on the simplicial complex as obtained via a convex hull algorithm. This is different from the triangulation of chordal graphs through variable elimination for exact inference. We would like to refer the interested reader to introductory texts on algebraic topology and homology [128, 140, 77].
where \( x_\kappa \) is the subset of variable nodes \([x(u), x(v), x(w)]\) that comprise the 2-simplex \( \kappa \).

More formally, a simplicial complex \( \Delta \) is a collection of simplexes of arbitrary dimensions with the following two properties:

1. For all simplexes \( \kappa \in \Delta \), if all faces \( \alpha \in \kappa \) then, \( \alpha \in \Delta \).

2. Any two simplexes \( \kappa_i, \kappa_j \in \Delta \) are either disjoint or intersect in a single whole face.

In other words, a simplicial complex is a geometrical structure that has a set of simplexes “glued” together along their faces. With these concepts in hand, we are able to describe the joint probabilities \( P_{\alpha \setminus \partial \Delta}(x(u), x(v)) \) in Equation (5.4) as the factor potentials on the edges in the interior of the factor graph in the central panel of Figure 5.2.

### 5.4.2 Simplicial Spanning Tree

It is straightforward to show that each interior edge of the grid-graph will appear twice in the joint distribution \( P_{\Delta}(x) \) of the simplicial complex. On the other hand, the edges at the boundary of the factor graph will appear only once in the quotient probability factorisation of the simplicial complex. As a result, the joint distribution of graph \( P(x) \) is the joint distribution for the collection of simplexes divided by the interior edges as described in Equation (5.4). This is depicted by the graph in the centre of Figure 5.2.

Equation (5.2) is a formulation of the nature of the exact inference of a junction tree. Exact inference is intractable for larger graphs with cycles for two reasons. Firstly, variable elimination to determine the chordal graph with the smallest treewidth\(^3\) is NP-hard [12]. Secondly, the cliques produced by variable elimination for many of the graphs often employed in practical problems are large, such that computing the permutations of the joint probability \( P(x_c) \) of such cliques is infeasible. This is even so since the complexity of a graph is exponential in size to the treewidth [44].

---

\(^3\)The treewidth of a tree-structured factor graph \( \mathcal{G} \) is the maximum number of unique variables from any of the cliques of the graph, i.e. \( w_G = \max_c |x_c| \).
In this chapter, we use an alternative tree-based algorithm that resembles the junction tree and which places constrains upon the clique construction so as to limit its treewidth. To this end, we connect the disjoint triangles of a simplicial complex into a simplicial spanning tree (SST) \([96, 45]\) as portrayed in the right-most panel of Figure 5.2. It is worth noting in passing that the number of SSTs in a simplicial complex can be determined from simplicial matrix-tree theorems \([96, 45]\). Here we construct our SST using maximum-spanning-tree algorithms (e.g \([142, 107]\)). The motivation for this is that inference on a SST is tractable and exact. Moreover, computations can be made in a two-pass fashion across all variables in the graph \(G\). We elaborate further on this in Section 5.4.4.

In addition, the reparameterised joint distribution approximates the exact Gibbs distribution in Equation (5.2). This reparameterised approximation of the Gibbs distribution is given by the tractable probability distribution in the spanning tree \(Q(x)\) as follows

\[
Q(x) = \frac{\prod_{\kappa \in K} P(x_{\kappa})}{\prod_{s \in S} P(x_s)}. \tag{5.6}
\]

The probabilities \(P(x_s)\) in Equation (5.6) represent the potentials at the edges connecting the cliques/simplexes on the graph. This implies that, as before \(x_s = x_{\kappa_u} \cap x_{\kappa_v}\) for any pair of adjacent simplexes \(x_{\kappa_u}, x_{\kappa_v}\).

Note that the adjacency of simplexes \(\kappa_u\) and \(\kappa_v\) is determined by the maximum-spanning-tree algorithm. The marginals for each of the variables \(x(u) \in x\) in the simplexes or cliques are recovered by marginalising the respective clique potentials \(P(x_{\kappa})\) after computing the distribution \(P(x)\).

For the spanning tree example shown on the right-hand panel of Figure 5.2 it becomes evident that the error in the approximated distribution \(P(x)\) is due to the duplicated variables appearing in the leaves and are omitted from the separator sets\(^4\). This is more evident by comparing the reparameterisation of Equation (5.6) to the exact Gibbs distribution of the graph in Equation (5.4). Note that the numerator is a direct factorisation of the cliques as expressed in

\(^4\)The distribution \(Q(x)\) computed from the simplicial spanning tree in the Figure 5.2(c) is
Figure 5.3: Example of the interactions across layers in a factor graph with variables shown as round markers and factors as squares.

Equation (5.5) whereas the probabilities for the factors on the interior edges is an approximation given by

\[
\prod_{(u,v)} P_{\alpha\beta\delta}(x(u), x(v)) \approx \prod_{s \in \mathcal{S}} P(x_s). \tag{5.8}
\]

5.4.3 Layered Factor Graphs

We now consider constructing a general graph comprised by a set of “layers”, whereby each of the nodes in these layers correspond to one of the algorithms given by

\[
Q(x) = \frac{P(x_1, x_2, x_4)P(x_2, x_4, x_5)P(x_2, x_5, x_6)P(x_2, x_3, x_6)}{P(x_2, x_4)P(x_2, x_5)P(x_2, x_6)P(x_4, x_5)} \times \frac{P(x_4, x_5, x_8)P(x_4, x_7, x_8)P(x_5, x_6, x_8)P(x_6, x_8, x_9)}{P(x_4, x_8)P(x_5, x_8)P(x_6, x_8)} = \frac{P(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9)}{P(x_6)}. \tag{5.7}
\]

Note the added terms with respect to the probability \(P(x)\) in Equation (5.3).
under consideration. In Figure 5.3, we show an example of a factor graph arising from such an evidence combining setting. In the figure, the factors are given by squared markers whereby the variables in the graph are denoted by circles on the image lattice. Note that, in this layered structure, the interaction between variables across methods is given by interlayer relationships.

From a probabilistic viewpoint, we can observe that, in Figure 5.3, interacting variables can be used to define the conditional distributions within each layer. More formally, let the factor graph $G$ be given by a set of subgraphs $G_\gamma$ for each of the layers $\gamma \in \mathcal{L}$ such that

$$G = \bigcup_{\gamma \in \mathcal{L}} G_\gamma.$$ 

We can express the probability of the hidden variable $x_\gamma(u) \in \Gamma$ for the $u^{th}$ token $\tau(u)$ using prototypes $\omega_i \in \Upsilon$ as follows

$$P(\Gamma|G) = \frac{1}{Z} \prod_{c \in C} \psi(\Gamma_c; G) = \frac{1}{Z} \prod_{u, v \in G} P(\tau(u), \tau(v)|x(u), x(v)) \prod_{\gamma \in \mathcal{L}} \prod_{\omega_i \in \Upsilon} P(x_\gamma(u)|\omega_i),$$

where $C$ is the clique-set of the graph, $u \sim v$ implies the nodes $u$ and $v$ are adjacent to each other, $\psi(\Gamma_c; G)$ is the potential function over the nodes corresponding to the hidden variables in the graph for the clique $c$ and $Z$ is a partition function.

It is worth noting in passing that Equation (5.9) effectively conditions the hidden variables to the potential functions for the variable nodes over the graph and the prototypes in $\Upsilon$. These can, hence, be viewed as an analogue of the mean-of-class prototypes used in the machine learning community [71]. Since each of the potential functions is defined over a subset of variables $\Gamma_c$ which comprises the estimates at each of the cliques $c \in C$, every variable in a clique is connected to at least one other variable through a factor node. The union of variables, factors, and the edges that connect them constitute the factor graph $G$ whose probability functions are dependant upon the variables $x(u)$, the tokens $\tau(u)$, the hidden variables $x_\gamma(u) \in \Gamma$, the prototypes $\omega_i \in \Upsilon$ and the relationships
between them.

In Equation (5.9) we have used Bayes’ rule to state the problem in terms of the likelihood and prior distributions with the partition function $Z$ acting as a normalisation constant. Here, we have also assumed that the priors $P(x_\gamma(u)|\omega_i)$ for our prototypes across layers $\gamma \in \mathcal{L}$ are independent from each other. Note that, in practice, the neighbours on each layer can be used to compute cliques which, in turn, can be employed to define the likelihood function for the inference process. This is reflected by the probability $P(\tau(u), \tau(v)|x(u), x(v))$, which captures the likelihood of the variables at neighbouring nodes given their respective beliefs. In the other hand, the priors $P(x_\gamma(u)|\omega_i)$ correspond to the likelihood of a belief for the $u^{th}$ token in layer $\gamma$ up to the prototype set $\Upsilon$.

Moreover, the Bayesian formulation in Equation (5.9) lends itself, by construction, to a probability factorisation over the graph. This is important since the inference in hand can now be posed as a maximum a posteriori (MAP) process whose aim of computation are marginal probabilities $P(x_\gamma(u))$. This can be viewed as the optimisation given by

$$\arg\max_{\Gamma} P(\Gamma|\mathcal{G}) = \arg\max_{\Gamma} \left\{ \prod_{u,v \in \mathcal{G}} P(\tau(u), \tau(v)|x(u), x(v)) \prod_{\gamma \in \mathcal{L}} \prod_{\omega_i \in \Upsilon} P(x_\gamma(u)|\omega_i) \right\} \quad (5.10)$$

which arises from Equation (5.9). Note that we have removed the partition function $Z$ from further consideration since it is a constant that does not affect the maximisation above.

### 5.4.4 Message Passing

If the probabilities $P(x_\gamma(u)|\omega_i)$ and $P(\tau(u), \tau(v)|x(u), x(v))$ can be computed, the MAP problem in Equation (5.10) becomes that of inferring the marginal probabilities $P(x_\gamma(u))$ of the variables at each layer. This can be achieved through the use of a sum/max-product algorithm [108].

Recall that the sum/max-product algorithm computes exact probability marginals of variables in a tree [138, 6]. In order to take advantage of this property, we partition our factor graph and convert the subgraphs into a collection of local tree
Figure 5.4: Simplicial spanning tree where clique variables (elliptical nodes) are connected to neighbouring cliques through separator sets (in parenthesis). The evidence-combined estimate from the geometric mean is denoted in squared braces.

structures. Note that performing exact inference on factor graphs with cycles, such as the grid-graph arising from a lattice, by converting them into trees is a procedure often established through variable elimination [196, 43]. These methods introduce fill-edges in lattice-like graphs, thus generating cliques with a large number of variables, i.e. with a large treewidth. As a result, the computational burden can potentially increase greatly when the clique potential is computed. Moreover, the large treewidth effectively draws many variables into a few cliques, changing the topology of the graph under study and potentially changing the inference problem in hand [15].

Here, we employ an alternative way to determine a variable elimination ordering using a convex hull algorithm for Delaunay triangulations [183]. Our choice stems from the fact that the Delaunay triangulations derived from the algorithm by Watson [183] form a simplicial complex is dependent only on the number and position of the nodes in the graph. This is important since the final simplicial complex is unique and independent of the order in which the nodes are processed during the triangulation procedure. Further, the computational complexity is proportional to the number of nodes \( n \), being not greater than \( O(n^2) \).

Figure 5.4 shows a simplified realisation of the SST resulting from the factor
graph for the MAP problem in Equation \((5.10)\). The separator set in the centre shows the joint interaction between variables across different layers. This is, effectively, a clique-tree realisation of the graph based on simplexes as shown in Figure \([5.3]\). In the figure, and for the sake of clarity, we have adopted the following notation. For a variable \(x(u)\) in layers \(\gamma, \phi\), we have used \(x_{\gamma,u}, x_{\phi,u}\). We also denote the cliques as \(c_i\) and the separators between a pair of adjacent cliques \(c_i, c_j\) as \(s_{ij}\). Where necessary, we assign an asterisk superscript to an updated joint distribution, \(i.e.\ P^\ast(x_{s_{ij}})\), with the number of asterisks indicating the number of times the distribution has been updated. Finally, we use the formulation in Equation \((5.6)\) to update the probabilities via division, where we use \(\mu\) to indicate a message flowing in the direction of the sub-indexed arrow.

Note that, from the figure, we can observe that the clique \(c_i\) “passes on” information on its corresponding probability to another clique \(c_j\) in two steps that describe the HUGIN algorithm \([8]\). The first of these updates the separator potentials, so that the message becomes

\[
\mu_{c_i \rightarrow s_{ij}}(x_{s_{ij}}) = \sum_{c_i \backslash s_{ij}} P(x_{c_i}), \tag{5.11}
\]

where \(P(x_{c_i})\) is the potential from all the variables in clique \(c_i\). To this end, we can view the potentials as products of the factors associated with the clique, \(i.e.\)

\[
P(x_{c_i}) = \prod_{x_c \subseteq x_{c_i}} \phi(x_c), \tag{5.12}
\]

where \(\phi(x_c)\) is the factor potential set by the variables \(x_c\). Equations \((5.11)\) and \((5.12)\) collectively model the algorithm in \([8]\).

The message in Equation \((5.11)\) then corresponds to the update of the separator potential, \(i.e.\ \mu_{c_i \rightarrow s_{ij}}(x_{s_{ij}}) = P^\ast(x_{s_{ij}})\). Note that before the updating procedure commences, the separator potentials are initialised by setting

\[
P(x_{s_{ij}}) = 1_\otimes, \tag{5.13}
\]

where \(1_\otimes\) denotes a tensor of ones whose dimensions are determined by the cardinality of the separator and the number of states of each variable.
Furthermore, the separator-to-clique interactions can be expressed explicitly as follows
\[ \mu_{s_{ij} \rightarrow c_j}(x_{c_j}) = \frac{P(x_{c_j})}{P(x_{s_{ij}})} \mu_{c_i \rightarrow s_{ij}}(x_{s_{ij}}). \] (5.14)

The message sent between two cliques \( c_i, c_j \) is, therefore, a two-step procedure that involves updating the separator potentials using Equation (5.11) so as to update the message-receiving clique potentials using Equation (5.14).

In Figure 5.2(c), we have illustrated the message passing process between neighbouring cliques through the separator sets. The message passing process entails updating probability distributions and marginalisation. It is important to note that the definition of the distributions at the cliques is determined by the nature of the factors. Moreover, we assume that the graph topologies in each layer are similar, such that the layer SSTs have a one-to-one correspondence between simplexes or cliques. To this end, we connect related simplexes via a “dummy” separator set of the evidence combining variables as shown in Figure 5.4. Further, we schedule messages in a way that interchangeably uses potentials from neighbouring layers can be used as priors to initiate the message passing process. For example, for the graph in Figure 5.4, messages can go from the leaves of layer \( \gamma \) towards the root and back in the opposite direction so that these potentials act as a priors to layer \( \phi \).

After the messages have been sent in both directions across all edges in the graph, each clique contains the updated posterior probability distributions of the variables to which the simplex is associated. In order to obtain the marginal distribution for any one variable \( x_\gamma(u) \) in the clique, we marginalise with respect to the adjacent variables making use of the joint distribution. The marginal of a given variable \( P(x_\gamma(u)) \) is obtained by using a sum/ maximum-rule so as to marginalise the posteriors over all other variables in the clique with respect to the one under consideration. This can be expressed as
\[ P(x_\gamma) = \sum_{x_{c_i} \setminus x_\gamma} P(x_{c_i}), \{ x_\gamma : x_\gamma \in x_{c_i} \}. \] (5.15)

As a result of this treatment, the estimate for the variable \( x_\gamma(u) \) at the token \( \tau(u) \) and layer \( \gamma \) may be computed using the posterior probability of the \( i^{th} \)
prototype $\omega_i$ given the belief $x_\gamma(u)$. This is since, for all the tokens under consideration this is equivalent to the distribution $P(\Gamma|G)$. Moreover, the estimate of the hidden variable at a layer $\gamma$ can be viewed as the expected value

$$x_\gamma(u) = \frac{\sum_{\omega_i \in \Upsilon} P(\omega_i|x_\gamma(u)) \omega_i}{\sum_{\omega_i \in \Upsilon} P(\omega_i|x_\gamma(u))},$$

(5.16)

such that the overall value of $x(u)$ can be approximated by the geometric mean,

$$x(u) = \left[ \prod_{\gamma \in \mathcal{L}} x_\gamma(u) \right]^{\mathcal{L}^{-1}},$$

(5.17)

where $\mathcal{L}$ is, as before, the set of layers in the factor graph $G$.

### 5.4.5 Prototype Matching Across Layers

Note that, so far, we have assumed that the prototype set for the graph is a global one that applies to the whole set of variable nodes in $G$. This is not always the case. Indeed the contribution of each of the methods under consideration to the prototype set $\Upsilon$ may be quite different, i.e. unrelated between layers. This is a problem since it cannot only greatly distort the geometric mean in Equation (5.17) but also lead to degenerate cases where the term

$$\prod_{\gamma \in \mathcal{L}} \prod_{\omega_i \in \Upsilon} P(x_\gamma(u)|\omega_i)$$

(5.18)

in Equation (5.10) is null.

To tackle this problem, we note that the variables $x_\gamma(u)$ across layers correspond to the same token $\tau(u)$. This suggests the prototypes contributing to $x_\gamma(u)$ in Equation (5.16) should be akin to each other across layers. This observation is important since it provides a notion of putative correspondence, i.e. an equivalence relation, between variables across different layers. Thus, here, we
employ an orthogonal linear map between layers such that

\[ x_\gamma(u) = K x_\phi(u) \]  (5.19)

where \( \phi \) and \( \gamma \) are two distinct layers in the graph.

The advantage of a linear map as the one above resides in the fact that this treatment yields a unique, i.e. Procrustean, transformation [85] which is invertible and can be computed in closed form via an SVD decomposition. It is worth noting in passing that this also lends itself naturally to further refinement via statistical approaches such as the EM algorithm [120] or relaxation labelling [38].

To see this more clearly, we employ Equation (5.16) so as to write the squared dissimilarity between variables on two layers as follows

\[
\epsilon = \sum_{u \in \mathcal{G}} \sum_{\gamma, \phi \in \mathcal{T}} \left( \frac{P(\omega_i | x_\gamma(u)) \omega_i}{\sum_{\omega_i \in \mathcal{T}_\gamma} P(\omega_i | x_\gamma(u))} - \alpha_{i,j} \frac{P(\omega_j | x_\phi(u)) \omega_j}{\sum_{\omega_j \in \mathcal{T}_\phi} P(\omega_j | x_\phi(u))} \right)^2
\]  (5.20)

where \( \alpha_{i,j} \) is the entry indexed \( i,j \) of the matrix \( K \), i.e. the linear map in Equation (5.19), \( \mathcal{T}_\gamma \in \mathcal{T} \) and \( \mathcal{T}_\phi \in \mathcal{T} \) are the prototype sub-sets for the layers \( \gamma \) and \( \phi \) in the graph.

Note that we can use the matrices \( M \) and \( D \) whose entries are given by

\[
M(i, u) = \frac{P(\omega_i | x_\gamma(u)) \omega_i}{\sum_{\omega_i \in \mathcal{T}_\gamma} P(\omega_i | x_\gamma(u))}
\]  (5.21)

and

\[
D(j, u) = \frac{P(\omega_j | x_\phi(u)) \omega_j}{\sum_{\omega_j \in \mathcal{T}_\phi} P(\omega_j | x_\phi(u))}
\]  (5.22)

and write the equation above in compact form as follows

\[
\epsilon = \| M - KD \|^2
\]  (5.23)

The expression in Equation (5.23) is important since the linear map \( K \) is, in fact, a Procrustean transformation over the variables \( x_\phi(u) \) for the set of tokens in layer \( \phi \). Recall that a Procrustean transformation is of the form \( Q = KD \), and minimises \( \| M - Q \|^2 \). It is known that minimising \( \| M - Q \|^2 \) is equivalent
Algorithm 6 - Evidence combining with prototype matching

Require: \( \tau \): tokens; \( \mathcal{L} = \bigcup \gamma \): methods which will correspond to factor graph layers

1: for all methods \( \gamma \in \mathcal{L} \) do
2: Construct SST \( \mathcal{T}_\gamma \) using Algorithm 7
3: for all \( \kappa \in \mathcal{T}_\gamma \) do
4: Compute initial potentials \( P(x_\kappa) \)
5: Update potentials using Equations (5.11) and (5.14)
6: end for
7: end for
8: Designate a reference layer using Equation (5.25)
9: Use probabilities to determine the linear map \( K \) (Equation (5.24))
10: Match prototypes to the base method using Equation (5.19)
11: Update simplex probabilities for the methods with a smaller entropy \( \phi \) using Equations (5.11) and (5.14)
12: for all \( u \) do
13: Compute \( x(u) \) using Equation (5.17)
14: end for
15: return \( x \): evidence combined variables

Moreover, let the singular value decomposition (SVD) of \( DM^T \) be \( USV^T \). It can be shown that the maximum of \( \text{Tr}[M(KD)^T] \) is achieved when \( V^TKU = I \), where \( I \) is the identity matrix. As a result, the optimal linear map \( K \) is given by

\[
K = VU^T.
\] (5.24)

5.5 Implementation

In Algorithm 6 we present the step sequence for our evidence combining approach as presented in Section 5.4. The algorithm takes the tokens and the methods which correspond to each of the factor graph layers at input and returns the variables \( x \) at output.

As mentioned earlier, we commence by constructing a simplicial spanning tree using Algorithm 7 for all the layers in our factor graph. These trees are computed across layers using the convex hull method in [133] and the maximum spanning tree algorithm in [142].
Algorithm 7 - Simplicial spanning tree (SST)

Require: $\tau$: tokens; factor graph layer $G_{\gamma}$
1: Assemble variables from factor graph into simplicial complex $\Delta$ using the convex hull method in [183].
2: Use the maximum spanning tree algorithm in [142] to connect simplexes $\kappa$.
3: for all $\kappa \in \Delta$ do
4: Formulate the factors $\phi$ from variables in $\kappa$.
5: Compute the initial potential $P(x_{\kappa})$ using Equations (5.12) and (5.13).
6: end for
7: return $SST \mathcal{T}$

For all the simplexes $\kappa$ in each of these trees, we update the potentials using Equations (5.11) and (5.14) (Line 2 in Algorithm 6). Note that, to make use of the linear map in Equation (5.19), one of the layers in the graph has to be “chosen” accordingly to have its beliefs mapped onto. We do this by computing all the potentials for each layer in Lines 4 and 5 of Algorithm 6. Once these are in hand, a reference layer $\gamma^*$ can be chosen using the max-entropy criterion given by

$$e_{\gamma^*} = \max_{\gamma,\phi} H(x_{c_{\gamma^*}}),$$  \hspace{1cm} (5.25)

such that $\ast$ denotes any of the layer indices $\gamma$ or $\phi$ and the entropy $H(x_{c_{\gamma^*}})$ of a given layer $\gamma$ is defined as

$$H(x_{c_{\gamma^*}}) = - \sum_{x_{\gamma} \in x_{c_{\gamma^*}}} \frac{P(x_{\gamma})}{\sum_{x_{\gamma}} P(x_{c_{\gamma^*}})} \log \left( \frac{P(x_{\gamma})}{\sum_{x_{\gamma}} P(x_{c_{\gamma^*}})} \right),$$  \hspace{1cm} (5.26)

where $c_{\gamma}$ denotes the root node of the corresponding simplicial spanning tree.

Once the reference layer has been determined, we match the prototypes of each layer to the designated reference accordingly in Line 10 of the algorithm and update the corresponding probabilities. Finally, the evidence-combined variables are computed using Equation (5.17).

5.6 Applications

In this section, we illustrate the utility of our factor graph for evidence combining on two applications. These are defogging and image segmentation.
5.6.1 Defogging

Defogging and dehazing are important image enhancement processes in object recognition, video surveillance, and segmentation. As a result, the topic has attracted considerable attention and interest in the image processing and computer vision communities.

Recall that, as the light travels in the atmosphere, haze and fog particles scatter the light. The extent of this scattering is a function of the distance the light travels, exhibiting an exponential decay in the reflected light transmission and an exponential increase in the airlight [133]. If we denote the image intensity at each pixel location \( u \) and wavelength \( \lambda \) as \( I(u, \lambda) \) we can express this behaviour using the expression

\[
I(u, \lambda) = L(\lambda)\rho(u, \lambda)e^{-\beta(\lambda)d(u)} + L(\lambda)(1 - e^{-\beta(\lambda)d(u)}),
\]

(5.27)

where \( d(u) \) is the depth variable, \( \rho(u, \lambda) \) is the pixel albedo, \( L(\lambda) \) is the skylight power spectrum, and \( \beta(\lambda) \) is a scattering coefficient which here we assume to be constant value of 1.75 across wavelengths [135].

5.6.1.1 Implementation

For defogging, we combine evidence provided by the methods in [135] and [167]. The first factor graph layer for the approach in [135] consists of a dual estimation procedure for the albedo and depth. To this end, Equation (5.27) above was rearranged as follows

\[
\tilde{I}(u, \lambda) = C(u, \lambda) + D(u, \lambda),
\]

(5.28)

where \( C(u, \lambda) = \ln (1 - \rho(u, \lambda)) \), \( D(u, \lambda) = -\beta(\lambda)d(u) \) and \( \tilde{I}(u, \lambda) = \ln \left( 1 - \frac{I(u, \lambda)}{L(\lambda)} \right) \) arises after applying the natural logarithm to the equations above.

Here, the aim is jointly estimating \( d(u) \) and \( \rho(u, \lambda) \) for all \( u \) and \( \lambda \) by maximising the posterior. Thus, the aim of computation becomes

\[
(d^*, \rho^*) = \arg\min_{d, \rho} - \ln P(C, D|\tilde{I}),
\]

(5.29)
where we have used bold font to imply that the corresponding literal accounts for all the variables of its kind, i.e. $\rho = \bigcup_{u,\lambda} \rho(u, \lambda)$. Note that the estimation problem in Equation [5.29] is a joint MAP one, where the independence between the albedo and depth as represented by two latent variables does not apply.

As such, for the method in [135] the structure of the subgraph is multilayered, involving the simultaneous estimation of the albedo and depth variables. Here, we elaborate further on the probabilities arising from the posterior probability in Equation [5.29] that gives rise to the priors imposed on the depth and albedo, i.e. $P(C, D | \tilde{I}) \propto P(\tilde{I} | C, D) P(C) P(D)$.

The prior on the albedo is hence obtained by fitting a generalised Gaussian distribution to the chromaticity gradients of the image. This yields

$$P(C) = \prod_{u,\lambda} \exp(\mathcal{V}_C),$$

where $\mathcal{V}_C$ is a smoothness energy term of the form

$$\mathcal{V}_C = \frac{|\rho(u, \lambda) - \rho(v, \lambda)|^\psi}{\eta},$$

where the parameters $\psi$ and $\eta$ are estimated using the method described in [164] and, as before, $v$ is in the neighbourhood of $u$. In addition, we use a depth prior that is dependent on the scene structure such that

$$P(D) = \prod_{u} \exp(\mathcal{V}_D),$$

where $\mathcal{V}_D$ is a potential function based upon a Laplace distribution. Our choice of distribution responds to the nature of real-world scenes, which contain both planar and smoothly varying regions. The variable $\mathcal{V}_D$ is given by

$$\mathcal{V}_D = \frac{|d(u) - d(v)|}{\varphi}. \quad (5.30)$$

Since the prior probabilities involve neighbouring pixels, we model them with
the Ising model \( [113] \) such that the energy function is

\[
- \ln P(C, D | \tilde{I}) = \sum_{u, \lambda} \left[ \frac{||\tilde{I}(u, \lambda) - C(u, \lambda) - D(u, \lambda)||^2}{2\sigma^2} + \sum_{u \sim v} (\mathcal{V}_C + \mathcal{V}_D) \right],
\]

(5.31)

where we have approximated the likelihood with a normal distribution, \( i.e. P(\tilde{I}|C, D) = \prod_{u, \lambda} \mathcal{N}(\tilde{I}, C, D, \sigma^2) \).

In contrast to the method in [135], the factor graph layer for the approach in [167] is concerned with estimating the airlight \( A(u) \), from the energy function

\[
E(A|p_u) = \sum_u \phi(p_u|A(u)) + \sum_{u, v \in N_u} \psi(A(u), A(v)),
\]

(5.32)

where \( p_u \) represents a small image patch centred at pixel \( u \), such that the airlight \( A(u) \) is constant. Note that the airlight here is wavelength independent and, using the expression in Equation (5.27), we have

\[
A(u) = \sum_{\lambda} L(\lambda) \left( 1 - e^{-\beta(\lambda)d(u)} \right).
\]

(5.33)

The likelihood term \( \phi \), measures the number of edges of the normalised clear image for a given airlight value [167], which yields

\[
\phi(p_u|A(u)) = \sum_{u, \lambda} |\nabla[\rho(\cdot, \lambda)]_{A(u)}|,
\]

(5.34)

where \( \rho(\cdot, \lambda)_{A(u)} \) is the albedo/clear image pixel intensities in the patch centred at \( u \) and for the airlight \( A \).

We explicitly express the normalised albedo as a function of airlight as follows,

\[
\rho(u, \lambda) = \left( \frac{I(u, \lambda)}{L(\lambda)} (\Sigma_{\lambda} L(\lambda)) - A(u) \right) e^{\beta(\lambda) d(u)}.
\]

(5.35)

As a result, for the layer corresponding to the method in [167], the airlight prior is similar to that in Equation (5.30) and is given by

\[
\psi(A(u), A(v)) = 1 - \frac{|A(u) - A(v)|}{\Sigma_{\lambda} L(\lambda)}.
\]

(5.36)
We would like to refer the interested reader to [167] for an in-depth derivation of the expressions above. The factor graph layer for this method can be defined as 

\[ P(x) = \frac{1}{Z} \exp(-E(x)) \]

using the Gibbs distribution in Equation (5.1), where the energy function is given by Equation (5.32).

Once the graph has been constituted as described above, we combine the airlight outputs, i.e. beliefs, yielded by the depth for the method in [135] and the airlight for the method in [167] making use of Equation (5.33) and the prototype matching procedure in Section 5.4.5.

### 5.6.1.2 Initialisation

Recall that Algorithm 6 requires an initial estimate for both layers. To initialise the inference process for the method in [135] over the corresponding layer of the factor graph, we note that, at each iteration, the MAP process is conducted over a discrete state space. We have done this following [135]. The state of either variables, i.e. the depth or the albedo, is updated so as to maximise the posterior.

For the initial estimate of the depth, we make use of the farthest possible distance estimate as computed from \( \tilde{I}(u, \lambda) \). This is, again, consistent with the initialisation used in [135], with the caveat that, since in our case we have a wavelength dependent scattering coefficient, the initial estimate becomes

\[ d_0(u) = \min_\lambda \frac{\tilde{I}(u, \lambda)}{-\beta(\lambda)}. \tag{5.37} \]

This contrasts with the method in [167], where we follow the initialisation of the airlight as presented by the authors. The initial airlight \( A_0 \) is derived from a blurred image based on the YIQ colour model estimate,

\[ A_0(u) = 0.257I'(u, \lambda_r) + 0.504I'(u, \lambda_g) + 0.098I'(u, \lambda_b), \]

where the pixel values at any of the red, green and blue colour channels for the transformed image \( I' \) are given by

\[ I'(u, \lambda) = \frac{I(u, \lambda)}{L(\lambda)} (\Sigma_\lambda L(\lambda)). \]
Figure 5.5: Defogging results. Top row: Example foggy images; Second row: Depth and albedo yielded by the method in [135]; Third row: Results delivered by the method in [167]; Bottom row: Depth and albedo results yielded by our evidence combining factor graph.

Note that for the two methods the skylight $L(\lambda)$ is assumed to occur at the horizon, i.e. at $d = \infty$. Its value is thus determined by finding a small patch with the largest mean intensity at each wavelength $\lambda$. The skylight is then given by

$$L(\lambda) = \frac{\sum_u I(p_u, \lambda)}{|I(p_u, \lambda)|}.$$

5.6.1.3 Results

In Figure 5.5 we illustrate the utility of our method for evidence combining in image defogging. To this end, we have obtained albedo and depth by combining the results yielded by the Bayesian defogging approach [135] and the contrast enhancing method in [167].

From the figure, it becomes evident that our method obtains a single depth and albedo building upon the strengths of both approaches. This can be observed in the resulting albedo, which is less saturated than that delivered by the method
in [167]. The depth recovered by our factor graph, on the other hand, is less prone to be affected by brightness and colour variations in the input images. These results are also in accordance with the notion that, whereas the airlight estimated by [167] is correlated to the colour, the method in [135] fails when large variations in depth occur. On the other hand, the albedo recovered by the method in [135] is less saturated. Thus, by combining the outputs of both methods, we are effectively taking advantage of their strengths to improve the defogging results.

5.6.2 Image Segmentation

Segmentation is one of the most common labelling problems encountered in areas such as computer vision, pattern recognition and image processing. Here, we illustrate the utility of our factor graph for purposes of colour image segmentation. Let each image consist of a given number of clusters \( k = 1, 2, ..., K \). Each cluster is represented by a mean colour value, \( \varsigma_k \). These mean colour values can be viewed as the prototypes introduced earlier. Thus, our goal of computation becomes that of assigning each pixel to a cluster, whereby we aim at recovering the cluster mean colour and the corresponding pixel labels.

5.6.2.1 Priors and potentials

For image colour segmentation, it is straightforward to define the likelihood probability \( P(\tau(u)|x(u)) \) as a multivariate Gaussian mixture model over the \( K \) classes as follows,

\[
P(\tau(u)|x(u)) = \sum_k \pi_k P(u|x_\gamma(u) = k),
\]

where \( \tau(u) \) in this case is the pixel colour and

\[
P(\tau(u)|x_\gamma(u) = k) = \frac{1}{\sqrt{2\pi|\Sigma_k|}} \exp \left( -\frac{1}{2} (\tau(u) - \mu_k)^T \Sigma_k^{-1} (\tau(u) - \mu_k) \right)\]
and the mixture prior is

\[ \pi_k = \sum_{\mathcal{N}_y} \exp \left( -\beta \left( \delta(x_{\gamma,k}(u) - x_{\phi,k}(u)) - 1 \right) \right) \] (5.39)

subject to \( \sum_k \pi_k = 1 \).

Note that, despite its simplicity, the treatment above has the advantage that classifying connected variables, i.e. pixels, across layers in the factor graph admits a wide variety of penalty functions. This is, indeed, a desirable trait which we exploit using the smoothness prior within layers given by

\[ P(x_{\gamma}(u)) = \sum_{v \in \mathcal{N}_u} \exp \left( -\alpha (x_{\gamma}(u) - x_{\gamma}(v))^2 \right) . \] (5.40)

5.6.2.2 Implementation

As per our defogging example application, in order to construct a multi-layered graph, we will make use of the image segmentation estimates from other methods. As a result, we first obtain an over-segmentation of the images using two existing methods, i.e. those in \cite{49} and \cite{40}.

Once the segmentation results for the two methods is in hand, we initialise the mean colours for each region yielded by the two methods. Moreover, the regions resulting from the segmented images are then used to construct the layered graph, each layer representing each of the two methods. To this end, we use the region centroids for each estimate and compute the Delaunay triangulation on the image plane for each of the segmentation images at our disposal. These triangulations are used for the layer edge-set of our factor graph where the connectivity across layers is constructed by first projecting region centroids for two neighbouring layers onto the image plane. A new Delaunay triangulation for the projected points is obtained so that an edge is considered to be across layers if the two connected points come from different segmentations.

With the factor graph in hand, we obtain the log-likelihood by combining
Equations (5.38), (5.39) and (5.40). This yields the following cost function

\[ -\arg\min_{x, \Theta} \log P(x | \tau, \Theta) = \]

\[ \arg\min \log(2\pi)|\Sigma| + \sum_{u} \left[ \frac{1}{2}(\tau(u) - \mu_k)^T \Sigma_k^{-1}(\tau(u) - \mu_k) + \beta \left( (\delta(||x_{\gamma,k}(u) - x_{\phi,k}(u)||) - 1) + \alpha ||x_{\gamma,k}(u) - x_{\gamma,k}(v)||^2 \right) \right], \]

where \( \Theta = \{\varsigma, \sigma\} \) is a set of parameters. Note that the term for the square root of \( 2\pi \) has been omitted from further consideration since it is constant and, hence, does not affect the minimum of the cost function. Also, in the equation above, \(|\Sigma_k|\) is the determinant of the diagonal matrix of the variance of cluster \( k \), \( \Sigma_k = \text{diag}\{\sigma_{ka}, \sigma_{kb}\} \).

Note that the two alternatives may yield segmentations that are very far apart in terms of the variable \( x_{\phi,k} \) in the mixture prior in Equation (5.39) above. Thus, we “match” these between layers using the linear map in Equation (5.19). This linear map is then used across layers i.e.

\[ x_{\gamma}(u) = K x_{\phi}(u) \]

when the message is passed from the layer with variables \( x_{\phi}(u) \) to that with variables \( x_{\gamma}(u) \). We make use of the inverse of the linear map when messages flow in the opposite direction.

### 5.6.2.3 Results

In figure 5.6, we illustrate the combination of the results yielded by both of the approaches used for our factor graph layers (the methods in [40] and [49]) into a single segmentation map. For the example image depicted in the figure, our factor graph method improves the segmentation by smoothing the sky based upon the evidence from the segmentation result delivered by the method in [49]. Information from the mean shift segmentation [40] is utilised by the factor graph to “smooth out” the hill against the background on the left-hand side of the image. This delivers the result in the right-hand panel of figure 5.6. Furthermore,
Figure 5.6: Segmentation results. Left-hand panel: Image from the BSDS500 dataset [125]; Central panels, from top-to-bottom: Segmentation results delivered by the methods in [49] and [40], respectively; Right-hand panel: Result yielded by our evidence combining factor graph.

Figure 5.7 compares the segmentation result from evidence combining against our earlier results in Figure 3.6. The image results confirm the improvement with smoother segmentation results.

To provide a more quantitative result regarding the combination of both methods, we have followed the methodology in [147] so as to compute, for both alternatives and our combined results, the precision-recall graph and the corresponding F1 scores. For our experiments, we have used the BSDS500 dataset [125], where the statistics for each image are obtained by computing the gradient of the segmentation map and matching the resulting boundaries to each of the five hand-labelled boundaries provided in the human ground-truth for the dataset. In the figure, we have also included the graph and F1 score for the low treewidth subtrees given in Figure 3.7.

In Figure 5.8, we show the precision-recall curves and F1 scores for our method and the alternatives. In the figure, we have also included the precision-recall curve and F1 score for low treewidth subtrees of Chapter 3. Note that, in the precision-recall curves, our method performs better than either of two alternatives alone. It also outperforms the results presented earlier in Figure 3.7. This is expected,
Figure 5.7: Comparative results. Left: Sample images from the BSDS500 dataset \[125\]. Middle: Low treewidth subtrees segmentation result from Chapter 3. Right: Segmentation result from our evidence combining approach.

since our method is effectively building upon the strength of \[49\] to merge smaller segments and the capacity of the mean shift method to obtain clusters that correspond to the mode of the colour distribution in the image. This, again, illustrates the benefit of combining methods that have different desirable properties through the use of factor graphs. Moreover, our method delivers a noticeable improvement
Figure 5.8: Performance of the evidence combining factor graph presented in this chapter compared with the low treewidth subtrees in Chapter 3 and the segmentation algorithms in [10, 192] and [19]. The plots are also measured against human ground-truth, following the methodology in [147].

over low treewidth subtrees with a comparable computational cost\(^5\).

### 5.7 Conclusion

In this chapter, we have presented an evidence combining method which exploits the ability of factors graphs to combine results from multiple algorithms. This yields an evidence combining scheme which is quite general in nature, being able to operate upon algorithms that are often used independently. We have shown how the sum/max-product algorithm can be used to compute the marginals over the graph by applying a Delaunay triangulation. We have also avoided degenerate cases making use of a Procrustean transformation between marginals. We have illustrated the utility of the method for purposes of evidence combining in defogging and colour image segmentation, where the combined methods clearly improve over the independent application of the alternatives under consideration.

\(^5\)Our experiments were performed on an Intel i7 at 2.7 Ghz with 8Gb of RAM. In the experiments, our inference process took 25.3 sec on average per image, whereas the low treewidth subtrees took approximately 22 sec. Both methods were implemented in Matlab.
Conclusion

The concluding chapter of this thesis reviews the main contributions from the research on factor graphs presented earlier. In addition, drawing from our conclusions we argue future directions that can be worthwhile to investigate. Therefore, in the first Section 6.1 of the chapter we will summarise the main contributions from Chapters 3, 4 and 5. Once the contributions are exhausted, Section 6.2 concludes this chapter with potential directions for future research.

6.1 Contributions

This thesis has used factor graphs in the fields of computer vision and image processing where their use has been limited. The research has particularly focused on instances where multiple variables or tokens interact in a complex way, either in layers, couplings, or multiple levels of scale. While existing graph-based methods are available in literature, by using factor graphs we have presented a unified representation for all layered graphical models. This transition from using directed or undirected graphical models to using factor graphs is one main contribution of this thesis. Moreover, this modelling is extendable to applications outside computer vision and image processing.

The factor graphs in this thesis have described probabilistic factorisations for MAP formulations in ways that have aided the implementation of inference algorithms. They have done so through the increased representational power they provide, by allowing us to specify the factorisation of the global function using factors. This has broadened our ability to design algorithms that are computationally efficient and achieve better inference results by exploiting this new structure. In the first instance in Chapter 3, we have partitioned regular and
irregular structured graphs into local subtrees that allow exact inference to be determined straightforwardly. Unlike in belief propagation where there is one message update equation, the direct likelihood and prior factorisations in the global function are utilised to give distinct message update equations. Further, the subgraphs have been constructed with Delaunay triangulations governing adjacency relationships in such a way that local exact inference is tractable. We have applied our algorithm to the segmentation and image enhancement tasks.

In Chapter 4, we demonstrated the efficacy of factor graphs in pixelwise illuminant estimation using multiple scales. As mentioned above, the use of factor graphs allowed us to utilise their specific factorisation. Using these additional associations allowed us to improve the inference, which was carried out on the entire graph. Moreover, throughout this thesis we have used Delaunay triangulations for various purposes. In Chapter 3 we have employed Delaunay triangulations to enforce adjacency constrains by adding indices one at a time. Instead, in Chapters 4 and 5 we engage the convex hull algorithm for added representation that better approximates the exact distribution. The associations across multiple scales in Chapter 4 for the colour constancy problem have improved single layer methods such as [19]. The chapter introduces the use of beliefs from other methods as priors and the ensembling of outputs from different methods using a geometric mean.

Lastly, in Chapter 5 we reparameterised the factor graph created by the convex hull algorithm into a spanning tree of simplexes. Recall that Chapter 3 uses Delaunay triangulations to define how variables are related to their neighbours. In Chapter 4 the convex hull algorithm for Delaunay triangulations has been used to extend the adjacency constraints provided in Chapter 3 to grid-structured graphs, and ensure connectivity and inference is performed globally. In Chapter 5 the convex hull allows for a tree reparameterisation that also spans all of the graph edges, and adds new associations in simplexes. This constrained the treewidth, while at the same time approximating the junction tree representation which computes exact inference. Furthermore, we have gone on to use beliefs from adjoining methods, or layers in the graph, to enforce consistency for combining evidence across layers with the help of Procrustes transformations.

Note that in Chapter 4 we combined outputs, but have not used beliefs nor
made use of the Procrustes transformation. Therefore, an additional contribution here has been to deal with degenerate cases and ensure prototypes across layers are consistent. The first step involved identifying the method with more degenerate cases for which we have relied on a measure of entropy. With an identified method in hand, its prototypes are matched using a kernel obtained from solving the orthogonal Procrustes problem. This produced a more elaborate and holistic solution to combining evidence from different methods. To our knowledge, the method in Chapter 5 is a novel approach for evidence combining which is not only applicable to computer vision tasks in this thesis, but is able to operate in other communities where different algorithms are often used independently.

### 6.2 Future Work

One can anticipate that the factor graph modelling framework presented here has great potential to be the solution for many future data-intensive problems that rely on inference and learning algorithms. This is since there has been a rapid rise of interrelated data that modern users and companies store, and would like to analyse. Furthermore, the ubiquitous use of the internet and social networks has also provided large amounts of data from which useful information would need to be extracted. Thus, data modelling and analytical tools will be increasingly vital for the future. Already, graphical models are driving powerful methods such as deep learning. These developments justify our belief that factor graphs will play a key role in providing the capabilities to solve future problems. In this thesis, we have demonstrated their power in the large data-centric field of computer vision. Besides the many possibilities of further research into applying factor graphs to pattern recognition and machine learning related fields, below we explore some future directions related to this thesis.

A large amount of the work done in this thesis has focused on linking two layers. However, other computer vision and more general problems involve complex systems that require greater representation. It would be useful to have more than two methods, for example, in evidence combining. Furthermore, exploring interactions derived from other layered representations and more complex associations indicated in the background Section 3.2 of Chapter 3 would be essential.
For example, one might be interested in adding a binary factor graph layer, or line process [64, 124], which preserves boundaries for the segmentation method in the chapter.

The local subtrees and simplicial spanning tree methods can be further analysed, regarding the quality of their approximation and convergence properties. The research can emphasise more on the computational benefits and accuracy derived from using the methods here compared to others in literature. As is the case in other analyses, we would expect to have a better insight into the nature of the error in our algorithms to be associated with the Bethe free energy. In particular, the error for the SST algorithm arises from the minimal spanning tree algorithm, where edges connecting simplexes determine the distribution in the denominator of the new parameterisation. This distribution should be as close as possible to the joint probabilities to be cancelled out in the numerator, so that the simplified joint probability corresponds to the joint distribution of the graph. It will then be easier to use convergence properties based on the Bethe free energy to draw conclusions on our algorithm. Another extension of the work presented here would be to relate the spanning trees to polytopes and matroids. Our novel reparameterisation in Chapter 5 constructs an $n$-dimensional polytope, so that it is possible to base one’s analysis on geometry and topology. From there, we may make assumptions that establish conjectures on the convexity, and existence of fixed points. This machinery can then be used to derive the sufficient conditions for convergence, and deduce the changes needed in reparameterisation that make it possible.

Finally, future investigations could also incorporate learning in carrying out MAP inference. In contrast to learning parameters in an unintelligible way, using factor graphs will be beneficial to the modelling as we will have a better understanding of how the intelligent network works. This will also help in the design and analysis of algorithms, and ultimately increase our capability to handle large scale datasets and computations.
A.1 List of publications


In this section, we demonstrate the equivalence between the direct sum-product algorithm given by Equations (2.18) and (2.19), and the corresponding junction tree Equations (2.26) and (2.27). Here, we aim to show that the inference from these equations yields equivalent exact marginals for variables in a tree-structured graph, using the example illustrations depicted in Figure 2.4.

B.1 Message passing for Figure 2.4(a)

To this end, we now provide an outline for the sum-product message passing steps in the first Figure 2.4(a) as directed by the numbered messages. Note that in the following, we express a handful of messages considered to be transmitted synchronously, and therefore assigned the same number in the message scheduling of Figure 2.4(a), to be called a batch.

Message batch 1:

\[
\begin{align*}
\mu_{x_1 \rightarrow f_{(12)}} &= 1_n \\
\mu_{x_3 \rightarrow f_{(23)}} &= 1_n \\
\mu_{f_{(4)} \rightarrow x_4} &= \psi(x_4) = P(x_4) \\
\mu_{f_{(5)} \rightarrow x_5} &= \psi(x_5) = P(x_5)
\end{align*}
\]
Appendix B

Message batch 2:

\[
\begin{align*}
\mu_{f(12) \rightarrow x_2} &= \sum_{x_1} \psi(x_1, x_2) \cdot \mu_{x_1 \rightarrow f(12)} = \sum_{x_1} P(x_2|x_1) \\
\mu_{f(23) \rightarrow x_2} &= \sum_{x_3} \psi(x_2, x_3) \cdot \mu_{x_3 \rightarrow f(23)} = \sum_{x_3} P(x_3|x_2) \\
\mu_{x_4 \rightarrow f(245)} &= \mu_{f(4) \rightarrow x_4} = P(x_4) \\
\mu_{x_5 \rightarrow f(245)} &= \mu_{f(5) \rightarrow x_5} = P(x_5)
\end{align*}
\]

Message batch 3:

\[
\begin{align*}
\mu_{x_2 \rightarrow f(245)} &= \mu_{f(12) \rightarrow x_2} \cdot \mu_{f(23) \rightarrow x_2} = \sum_{x_1, x_3} P(x_2|x_3)P(x_3|x_2) \\
\mu_{f(245) \rightarrow x_2} &= \sum_{x_4, x_5} \psi(x_2, x_4, x_5) \cdot \mu_{x_4 \rightarrow f(245)} = \sum_{x_4, x_5} P(x_2|x_4, x_5)P(x_4)P(x_5)
\end{align*}
\]

Message batch 4:

\[
\begin{align*}
\mu_{x_2 \rightarrow f(12)} &= \mu_{f(23) \rightarrow x_2} \cdot \mu_{f(245) \rightarrow x_2} = \sum_{x_3, x_4, x_5} P(x_3|x_2)P(x_2|x_4, x_5)P(x_3|x_2)P(x_5) \\
\mu_{x_2 \rightarrow f(23)} &= \mu_{f(12) \rightarrow x_2} \cdot \mu_{f(245) \rightarrow x_2} = \sum_{x_1, x_4, x_5} P(x_2|x_1)P(x_2|x_4, x_5)P(x_4)P(x_5) \\
\mu_{f(245) \rightarrow x_4} &= \sum_{x_2, x_5} \psi(x_2, x_4, x_5) \cdot \mu_{x_2 \rightarrow f(245)} \cdot \mu_{x_5 \rightarrow f(245)} \\
&= \sum_{x_1, x_2, x_3, x_5} P(x_2|x_1, x_5)P(x_2|x_1)P(x_3|x_2)P(x_5) \\
\mu_{f(245) \rightarrow x_5} &= \sum_{x_2, x_4} \psi(x_2, x_4, x_5) \cdot \mu_{x_2 \rightarrow f(245)} \cdot \mu_{x_4 \rightarrow f(245)} \\
&= \sum_{x_1, x_2, x_3, x_4} P(x_2|x_1, x_5)P(x_2|x_1)P(x_3|x_2)P(x_4)
\end{align*}
\]
Message batch 5:

\[
\begin{align*}
\mu_{f(12) \to x_1} &= \sum_{x_2} \psi(x_1, x_2) \cdot \mu_{x_2 \to f(12)} \\
&= \sum_{x_2, x_3, x_4, x_5} P(x_2|x_1) P(x_3|x_2) P(x_4) P(x_4) P(x_5) \\
\mu_{f(23) \to x_3} &= \sum_{x_2} \psi(x_2, x_3) \cdot \mu_{x_2 \to f(23)} \\
&= \sum_{x_1, x_2, x_4, x_5} P(x_3|x_2) P(x_2|x_1) P(x_2|x_4, x_5) P(x_4) P(x_5) \\
\mu_{x_4 \to f(4)} &= \mu_{f(245) \to x_4} = \sum_{x_1, x_2, x_3, x_5} P(x_2|x_4, x_5) P(x_2|x_1) P(x_3|x_2) P(x_5) \\
\mu_{x_5 \to f(5)} &= \mu_{f(245) \to x_5} = \sum_{x_1, x_2, x_3, x_4} P(x_2|x_4, x_5) P(x_2|x_1) P(x_3|x_2) P(x_4)
\end{align*}
\]

The algorithm terminates when the final messages towards the leaves are computed, i.e. once each edge in the graph has had a single message pass in both directions. After the sum-product of probabilities has terminated, the marginals are computed as follows using Equation (2.20):

\[
\begin{align*}
P(x_1) &= \mu_{f(12) \to x_1} = \sum_{x_2, x_3, x_4, x_5} P(x_2|x_1) P(x_3|x_2) P(x_2|x_4, x_5) P(x_4) P(x_5) \\
P(x_2) &= \mu_{f(12) \to x_2} \cdot \mu_{f(23) \to x_2} \cdot \mu_{f(245) \to x_2} \\
&= \sum_{x_1} P(x_2|x_1) \cdot \sum_{x_3} P(x_3|x_2) \cdot \sum_{x_4, x_5} P(x_2|x_4, x_5) P(x_4) P(x_5) \\
P(x_3) &= \mu_{f(23) \to x_3} = \sum_{x_1, x_2, x_4, x_5} P(x_3|x_2) P(x_2|x_1) P(x_2|x_4, x_5) P(x_4) P(x_5) \\
P(x_4) &= \mu_{f(245) \to x_4} \cdot \mu_{f(4) \to x_4} = \sum_{x_1, x_2, x_3, x_5} P(x_2|x_4, x_5) P(x_2|x_1) P(x_3|x_2) P(x_5) \cdot P(x_4) \\
P(x_5) &= \mu_{f(245) \to x_5} \cdot \mu_{f(5) \to x_5} = \sum_{x_1, x_2, x_3, x_4} P(x_2|x_4, x_5) P(x_2|x_1) P(x_3|x_2) P(x_4) \cdot P(x_5)
\end{align*}
\]
B.2 Message passing for Figure 2.4(b)

We now seek to obtain the same marginals using an equivalent junction tree representation for the graph topology in Figure 2.4(a). We will use the junction tree in Figure 2.4(b), which is a reparameterisation of the graph in Figure 2.4(a) as stated by Equation (2.25). Using the factorisation in Equation (2.25), the initial clique potentials of the tree can be computed as follows,

\[
\Psi(x_{c_1}) = \psi(x_1, x_2) = P(x_2|x_1) \\
\Psi(x_{c_2}) = \psi(x_2, x_3) = P(x_3|x_2) \\
\Psi(x_{c_3}) = \psi(x_2, x_4, x_5) \cdot \psi(x_4) \cdot \psi(x_5) = P(x_2|x_4, x_5)P(x_4)P(x_5).
\]

Message passing is conducted, as soon as the clique potentials are initialised, using Equation (2.27) as detailed in the inference procedure provided below. Here, we follow the scheduling enforced by the chosen root node, clique \(c_2\), in Figure 2.4(b). Furthermore, note that if there are no neighbouring messages present we assume the message vector to be unity.

Message 1:

\[
\mu_{c_1 \rightarrow c_3}(x_2) = \sum_{x_1} P(x_2|x_1)
\]

Message 2:

\[
\mu_{c_3 \rightarrow c_2}(x_2) = \sum_{x_4, x_5} P(x_2|x_4, x_5)P(x_4)P(x_5) \cdot \mu_{c_1 \rightarrow c_3}(x_2) \\
= \sum_{x_4, x_5} P(x_2|x_4, x_5)P(x_4)P(x_5) \cdot \sum_{x_1} P(x_2|x_1)
\]

Message 3:

\[
\mu_{c_2 \rightarrow c_3}(x_2) = \sum_{x_3} P(x_3|x_2)
\]
Message 4:

$$\mu_{c_3 \rightarrow c_1}(x_2) = \sum_{x_4, x_5} P(x_2|x_4, x_5)P(x_4)P(x_5) \cdot \mu_{c_2 \rightarrow c_3}(x_2)$$

$$= \sum_{x_4, x_5} P(x_2|x_4, x_5)P(x_4)P(x_5) \cdot \sum_{x_3} P(x_3|x_2)$$

Finally, akin to the traditional sum-product method, the clique beliefs are computed based on Equation (2.28) to give,

$$B(x_{c_1}) = \Psi(x_{c_1}) \cdot \mu_{c_3 \rightarrow c_1}(x_2)$$

$$= P(x_2|x_1) \cdot \sum_{x_4, x_5} P(x_2|x_4, x_5)P(x_4)P(x_5) \cdot \sum_{x_3} P(x_3|x_2)$$

$$B(x_{c_2}) = \Psi(x_{c_2}) \cdot \mu_{c_3 \rightarrow c_2}(x_2)$$

$$= P(x_3|x_2) \cdot \sum_{x_4, x_5} P(x_2|x_4, x_5)P(x_4)P(x_5) \cdot \sum_{x_1} P(x_2|x_1)$$

$$B(x_{c_3}) = \Psi(x_{c_3}) \cdot \mu_{c_1 \rightarrow c_3}(x_2) \cdot \mu_{c_2 \rightarrow c_3}(x_2)$$

$$= P(x_2|x_4, x_3)P(x_4)P(x_5) \cdot \sum_{x_1} P(x_2|x_1) \cdot \sum_{x_3} P(x_3|x_2) \quad \text{(B.2)}$$

It is then easy to “read-off” any marginal for a particular variable using any belief in which the variable appears, for which the reading can be verified to be equivalent to the marginal from Equation (B.1).
C.1 Reparameterisation of a junction tree

Here, we illustrate how a junction tree can be reparameterised as a quotient of the joint probabilities at the cliques and the distributions on the separator sets at its edges as given by Equation (5.2). To this end, we start with the marginal of the Gibbs measure in Equation (5.1) and write the joint distribution \( P(x_{c_i}) \) for a clique \( c_i \) as follows

\[
P(x_{c_i}) = \frac{1}{Z} \psi(x_{c_i}) \sum_{x_{c_j} \setminus x_{c_i}} \prod_{c_j \in C, c_j \neq c_i} \psi(x_{c_j}).
\]  

(C.1)

We now make use of the expression in Equation (C.1) to provide a proof by induction. Thus, consider Equation (5.2) for two cliques \( c_1, c_2 \) joined together by a maximum spanning tree (MST), i.e.

\[
P(x) = \frac{P(x_{c_1})P(x_{c_2})}{P(x_{s_{1,2}})},
\]

where \( s_{1,2} = x_{c_1} \cap x_{c_2} \) denotes the separator set between \( c_1 \) and \( c_2 \).

We can write the probability distributions at the two cliques as follows

\[
P(x_{c_1}) = \frac{1}{Z} \psi(x_{c_1}) \sum_{x_{c_2} \setminus x_{c_1}} \psi(x_{c_2})
\]

\[
P(x_{c_2}) = \frac{1}{Z} \psi(x_{c_2}) \sum_{x_{c_1} \setminus x_{c_2}} \psi(x_{c_1}).
\]  

(C.2)

Further, note that, by definition, the complement of the separator set for all
variables $x$ in this tree can be written as

$$x_{c_2} \setminus x_{c_1} = x_{c_1} \setminus x_{c_2} = (x_{c_1} \cup x_{c_2}) \setminus x_{s_{1,2}}, \quad (C.3)$$

By multiplying the two expressions in Equation (C.2) we get

$$P(x_{c_1})P(x_{c_2}) = \frac{1}{Z^2} \psi(x_{c_1})\psi(x_{c_2}) \sum_{x_{c_2} \setminus x_{c_1}} \psi(x_{c_2})\psi(x_{c_1})$$

$$= P(x) \sum_{(x_{c_1} \cup x_{c_2}) \setminus x_{s_{1,2}}} \frac{1}{Z} \psi(x_{c_2})\psi(x_{c_1})$$

$$= P(x) \sum_{(x_{c_1} \cup x_{c_2}) \setminus x_{s_{1,2}}} P(x)$$

$$= P(x)P(x_{s_{1,2}}). \quad (C.4)$$

where we have used the Gibbs measure (5.1) given by

$$P(x) = \frac{1}{Z} \prod_{c \in C} \psi(x_c)$$

$$= \frac{1}{Z} \psi(x_{c_1})\psi(x_{c_2}).$$

and marginalised over the variables in the separator set $P(x_{s_{1,2}})$.

Suppose now that the junction tree is comprised of $n$ cliques. This naturally yields $n$ equations, each of which accounts for the probabilities of the cliques in the tree, as expressed in Equation (C.1). The product of these equations is hence given by

$$\prod_{c_i \in C} P(x_{c_i}) = \frac{1}{Z^n} \prod_{c_i \in C} \left\{ \psi(x_{c_i}) \sum_{x_{c_j} \setminus x_{c_i}} \prod_{c_j \in C, c_j \neq c_i} \psi(x_{c_j}) \right\}$$

$$= \frac{1}{Z} \prod_{c_i \in C} \psi(x_{c_i}) \sum_{(x_{c_i} \cup x_{c_j}) \setminus x_{s_{ij}}} \left( \frac{1}{Z} \prod_{c_j \in C} \psi(x_{c_j}) \right)^{n-1}$$

$$= P(x) \sum_{(x_{c_i} \cup x_{c_j}) \setminus x_{s_{ij}}} [P(x)]^{n-1}$$
Reparameterisation of a junction tree

\[ P(x) \prod_{s_{ij}} P(x_{s_{ij}}), \]  

(C.5)

where \((x_{c_i} \cup x_{c_j})\) denotes that cliques \(c_i, c_j\) that are connected by an edge.

Moreover, it is important to note that the set \(|x_{c_i} \cap x_{c_j}| \geq 1\), i.e. the separator, is non-empty. For the second equality in the expression above, we have used the commutativity of the products over marginals so that, by interchanging indices \(i, j\), the product can be written as follows

\[
\prod_{c_i \in \mathcal{C}} \prod_{c_j \in \mathcal{C}} \psi(x_{c_j}) = \prod_{c_i \in \mathcal{C}} \prod_{c_j \in \mathcal{C}} \psi(x_{c_j}) \]

\[
= \prod_{c_i \in \mathcal{C}} \prod_{c_j \in \mathcal{C}} \psi(x_{c_j}) \]

\[
= \left( \prod_{c_j \in \mathcal{C}} \psi(x_{c_j}) \right)^{n-1},
\]

where the last equality is a repetition of the Gibbs distribution \(n - 1\) times. The third and fourth equalities in Equation (C.5) follow by definition. Note that Equation (C.5) is equivalent to Equation (5.2) once the indices \(i, j\) and \(n\) are modified accordingly. ■


Int. Wkshp Visual Surveillance and Performance Evaluation of Tracking and Surveillance, 301–308. (cited on page 72)


174. **Tukey, J. W., 1977.** *Exploratory Data Analysis*. Addison-Wesley. (cited on page 84)


