Integration and Goal-Guided Scheduling of Bottom-Up and Top-Down Computing Processes in Hierarchical Models

A dissertation submitted in partial satisfaction of the requirements for the degree
Doctor of Philosophy in Statistics

by

Tianfu Wu

2011
The dissertation of Tianfu Wu is approved.

Luminita Vese

HongQuan Xu

YingNian Wu

Song-Chun Zhu, Committee Chair

University of California, Los Angeles

2011
To my parents and my love,
for their continuous and unconditional support
# Table of Contents

1 Introduction ......................................................... 1
   1.1 Motivation and Objectives .................................. 1
   1.2 Intuitions of Scheduling ................................. 4
   1.3 Thesis Outline ........................................... 5
   1.4 Contributions ............................................ 6

2 Hierarchical AoG Representation and BU/TD Processes ........ 8
   2.1 Hierarchical AoG Representation ....................... 8
   2.2 Definition of an AoG .................................. 10
      2.2.1 Parse graph ...................................... 11
      2.2.2 Configuration ................................... 13
   2.3 The BU/TD Computing Processes Identified in an AoG ...... 13
      2.3.1 Background ..................................... 13
      2.3.2 Definitions of the $\alpha$-$\beta$-$\gamma$ Processes ....... 14

3 Evaluating Information Contributions of BU/TD Processes ..... 20
   3.1 Isolating BU/TD Computing Processes ................... 21
      3.1.1 Motivation ....................................... 21
      3.1.2 The Isolating Method ............................ 21
   3.2 Generating Training Data for BU/TD Computing Processes .. 23
   3.3 Learning BU/TD Computing Processes by MLE ............. 26
      3.3.1 Learning the $\alpha$ Process ...................... 26
3.3.2 Learning the $\beta$ Process ........................................... 30
3.3.3 Learning the $\gamma$ Process ........................................... 33
3.4 Evaluation Method ......................................................... 35
3.5 Human Study ............................................................... 37
  3.5.1 The Experimental Environment for Human Study .............. 37
  3.5.2 The Testing Data Used in Human Study ......................... 37
3.6 Experiments ............................................................... 38
  3.6.1 Junctions and Rectangles in Low-to-Middle-Level Vision .. 38
  3.6.2 Human faces in High-Level Vision ............................... 41

4 Integrating BU/TD Processes by Breadth-First-Search (BFS) 44
  4.1 Bayesian Formulation ................................................... 45
    4.1.1 The Prior Probability ............................................. 45
    4.1.2 The Likelihood Model ............................................. 46
  4.2 Object Parsing by Bayesian Inference ............................... 47
    4.2.1 Factorizing the Bayesian Posterior ............................. 47
    4.2.2 Connecting the $\alpha$-$\beta$-$\gamma$ Processes to Bayesian Inference ... 50
  4.3 Integrating the $\alpha$-$\beta$-$\gamma$ Processes by BFS ............. 56
    4.3.1 Connection with Dynamic Programming (DP) .................... 59
  4.4 Experiments ............................................................. 59
    4.4.1 Performance Comparisons ....................................... 59
    4.4.2 Rectangle Parsing ............................................... 60
    4.4.3 Human Face Parsing ............................................... 60
5 Learning Near-Optimal Cost-Sensitive Decision Policies . . . . 64
5.1 Background .................................................. 65
5.2 Problem Formulation ........................................ 67
  5.2.1 Factorizing a Scoring Function .......................... 67
  5.2.2 The Definition of a Decision Policy ...................... 68
  5.2.3 The Data Flow Distributed by a Decision Policy ...... 72
  5.2.4 The Risk Function of a Decision Policy ................. 76
5.3 Minimizing an Upper Bound of the Risk Function by DP ...... 80
5.4 Searching the Near-Optimal Decision Policy Given Allowable Accuracy Performance Bounds .................. 83
5.5 The Risk of a Testing Example after Computed ............... 84
5.6 Experiments .................................................. 85
  5.6.1 Learning the decision policy for a terminal node ....... 85
  5.6.2 Learning the decision policy for an And-node ........... 89

6 Scheduling BU/TD Processes by Best-First Heuristic Search . 90
6.1 Background .................................................. 91
6.2 Overview of Goal-Guided Scheduling .......................... 95
6.3 Heuristics of the \( \alpha \) Computing Process Using the Learnt Near-Optimal Decision Policy ......................... 97
6.4 Learning Context-Sensitive Decision Policy ................... 98
  6.4.1 Factorizing the Scoring Function of \( \alpha \) Process within a Parse Graph ................................. 98
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.4.2</td>
<td>The Context-Sensitive Decision Policy Learnt from Redis-distributed</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td>Data by Parse Graphs</td>
<td></td>
</tr>
<tr>
<td>6.5</td>
<td>Heuristics for Exploring Parse Graphs</td>
<td>101</td>
</tr>
<tr>
<td>6.6</td>
<td>The Scheduling Space</td>
<td>101</td>
</tr>
<tr>
<td>6.7</td>
<td>The Agenda-Chart Scheduling Algorithm</td>
<td>103</td>
</tr>
<tr>
<td>6.8</td>
<td>Summary</td>
<td>106</td>
</tr>
<tr>
<td>7</td>
<td>Conclusions and Discussions</td>
<td>107</td>
</tr>
<tr>
<td>7.1</td>
<td>Thesis Summary</td>
<td>107</td>
</tr>
<tr>
<td>7.2</td>
<td>Representational and Computational Knowledge</td>
<td>108</td>
</tr>
<tr>
<td>7.3</td>
<td>Future Work</td>
<td>109</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Motivation of the identified three types of computing process (BU/TD) using face detection as an example. ................. 2

2.1 Illustration of the And-Or graph (AoG) representation. ........ 9
2.2 Illustration of the $\alpha$-$\beta$-$\gamma$ computing processes. .............. 15
2.3 Illustration of interactions between the $\alpha$-$\beta$-$\gamma$ processes in an AoG 17
2.4 Examples of the identified $\alpha$-$\beta$-$\gamma$ processes for human faces. ..... 18

3.1 Illustration of isolating the $\alpha$, $\beta$ and $\gamma$ processes. .......... 22
3.2 Illustration of learned $\alpha$, $\beta$ binding and $\gamma$ prediction models for human face. ................................. 27
3.3 Illustration of the AoG for junctions and rectangles. ................. 28
3.4 Illustration of evaluating the information contributions of the $\alpha$, $\beta$ and $\gamma$ processes individually based on their discriminative power. 36
3.5 Plots of the information contributions of the $\alpha$ and $\beta$ processes of junctions and rectangles. ....................... 39
3.6 Plots of the information contributions of the $\alpha$ processes of nodes (i.e., head-shoulder, face, left eye, right eye, nose and mouth) in human face AoG in the human study. .......................... 40
3.7 Plots of the information contributions of the $\beta$ processes of the human face. ................................................. 41
3.8 Some examples used in the human study of evaluating the information contribution of the $\beta$ processes of human face. ......... 42
4.1 A running example of pursuing human faces and constructing corresponding parse graphs by integrating the \( \alpha, \beta \) and \( \gamma \) processes. 51

4.2 Illustration of constructing parse graphs by integrating the \( \alpha-\beta-\gamma \) processes: (a) and (b). 52

4.2 Illustration of constructing parse graphs by integrating the \( \alpha-\beta-\gamma \) processes: (c) and (d). 53

4.2 Illustration of constructing parse graphs by integrating the \( \alpha-\beta-\gamma \) processes: (e) and (f). 54

4.3 A running example of pursuing junctions and rectangles in a typical image by integrating the \( \alpha, \beta \) and \( \gamma \) processes. 61

4.4 Results of rectangle parsing and the ROCs. 62

4.5 Results of human face parsing and the ROCs. 63

5.1 Illustration of the decision policy. 70

5.2 Illustration of the paths of examples traversing in the decision policy. 71

5.3 Illustration of band width of the paths of examples traversing in the decision policy. 72

5.4 Illustration of selecting the allowable bounds on accuracy performance for a learnt computing process. 76

5.5 Learnt decision policies for the \( \alpha \) process of human faces using Adaboost classifier with different pairs of FP and FN loss costs used. 86

5.6 Another example of the learnt decision policies for human face Adaboost classifier. 87

5.7 Learnt decision policies for the deformable part model of an And-node using pedestrian as an example. 88
5.8 Another example of the learnt decision policy for pedestrian de-
formable part model. ....................................................... 89

6.1 Illustration of the scheduling space. ................................. 102
List of Tables

2.1 The table of notations for defining AoG and $\alpha$-$\beta$-$\gamma$ processes . . . 19
I would like to convey my utmost gratitude to my advisor, Professor Song-Chun Zhu, for all his help guiding me through my graduate study and research at UCLA. With his great vision, he has been encouraging, supportive and patient of my research, from the very beginning to end. I learned from him that doing good research by working hard with my heart is not only inspiring and exciting, but also enjoyable and satisfying, which will definitely keep lighting me on every aspect in my future life.

I would also like to thank my other thesis committee members, Professor Yingnian Wu, Professor Luminita Vese and Professor Hongquan Xu. I thank Professor Yingnian Wu for his always helpful and insightful discussions during my research. I thank Professor Luminita Vese and Professor Hongquan Xu for taking their valuable time to review my thesis and give helpful suggestions. I am really lucky to have these incredibly knowledgeable and kind committee members.

I would also like to thank other researchers with whom I worked together during my research: Xiong Yang, Guisong Xia, Hongwei Li, Ruxin Gao, Dengxin Dai, Jun Zhu, Quan Zhou and Dan Xie. Xiong Yang and I worked together on evaluating the information contributions of bottom-up/top-down computing processes. Guisong Xia, Hongwei Li and I worked together on the compositional boosting. Ruxin Gao and I worked together on computing the 2.1D sketch by mixed Markov Random Field. Dengxin Dai and I worked together on the unsupervised scene categorization. Jun Zhu and I worked together on the tangram model for scene configurations. Quan Zhou and I worked together on scene parsing by data-driven cluster sampling. Dan Xie and I worked together on learning the decision policy. I appreciate all of them as good friends and terrific teammates for their sharing both happiness and “suffering” during research and life.
I also feel very grateful for their helps during my study at UCLA to Wenze Hu, Xiaoyu Xu, Zhenyu Yao, Qiongchen Wang, Zhangzhang Si, Brandon Rothrock, Haifeng Gong, Jianting Wen, Mingtian Zhao, Kent Shi, Jake Porway, Liang Lin, Lin Nie, Yibiao Zhao, Shuo Wang, Mingtao Pei, Zhi Han, Yuanhao Chen, Jungseock Joo, Seyoung Park, Amy Morrow, Maria Pavlovskaia, Roozbeh Mottaghi, Xuming He, Shuang Wu, Caiming Xiong, Xingyao Ye and Chuanliang Chen. My gratitude also goes to all of my colleagues at Lotus Hill Research Institute (LHI), China, especially to Wenhua Xia and Huilan Zhu for their taking care of my daily life when I worked at LHI.

I would also like to thank all staffs in statistics department who are supportive whenever I went into their office with my problems.

I would also like to thank my M.S. advisor, Professor Jun Gao, at HeFei University of Technology (HFUT), China for his support during my study at HFUT. I would also like to thank Professor Chuanfang Dai at University of Science and Technology of China (USTC) for his support during my study at USTC. I also feel very thankful for their helps during my study at HFUT and USTC to Weibing Cao, Song Deng, Canju Lu, Zhao Xie, Xiaojia Wang, Bo Pang, Ping Zhang and Qing Zhao.

Last but not least, I thank my family for their love, sacrifice and unconditional support. I thank my girl friend, Xi Song, for her love and understanding.
Vita

1997–1999 Two-year College (Electronic Engineering), University of Science and Technology of China (USTC), China

2002–2005 M.S. (Computer Science), HeFei University of Technology (HFUT), China

2003–2005 Research Assistant, Institute of Intelligent Machines, Chinese Academy of Science, Hefei, China

2005–2008 Research Assistant, Lotus Hill Research Institute (LHI), China

2008–2010 Teaching Assistant, Department of Statistics, UCLA.

2008–2011 Graduate Student Researcher, Department of Statistics, UCLA.

Publications


Xiong Yang, Tianfu Wu and Song-Chun Zhu, “Evaluating Information Contributions of Bottom-up and Top-down Processes”, in *Proceedings of International Conference on Computer Vision (ICCV)*, 2009


Abstract of the Dissertation

Integration and Goal-Guided Scheduling of Bottom-Up and Top-Down Computing Processes in Hierarchical Models

by

Tianfu Wu

Doctor of Philosophy in Statistics

University of California, Los Angeles, 2011

Professor Song-Chun Zhu, Chair

Accuracy performance and computational efficiency are two of the most important issues of object detection and parsing in computer vision, and the trade-off between them is usually guided by vision goals. In the literature, to improve accuracy performance, hierarchical models have been widely and successfully used, but often at the expense of increasing the computational burden largely. The explosion of computing costs would practically prevent a computer vision system from scaling to hierarchical models which consist of a large number of nodes. Meanwhile, hierarchical models are studied with zero-one loss used for nodes (i.e., loss cost-insensitive).

The goal of this thesis is to present a framework of integrating and scheduling bottom-up (BU) and top-down (TD) computing processes in a recursively defined hierarchical And-Or graph (AoG) to address, with a numerical study, the vision-goal-guided trade-off between accuracy performance and computational efficiency in both loss cost-insensitive and cost-sensitive situations.

The BU/TD computing processes consist of three types of processes identified
for each node $A$ in an AoG: (i) The $\alpha(A)$ process detects node $A$ directly based on image features; (ii) The $\beta(A)$ process computes node $A$ by binding its child node(s) bottom-up; and (iii) The $\gamma(A)$ process predicts node $A$ top-down from its parent node(s). To evaluate their individual information contributions, the three processes are isolated and then trained separately. The learning of the three processes are formulated under the maximum likelihood estimation (MLE) framework. A numerical study of the information contribution is presented with both computer and human experiments. The experimental results show that the three processes contribute to computing node $A$ from images in complementary ways in terms of scale and occlusion conditions.

To improve the accuracy performance, the $\alpha$-$\beta$-$\gamma$ computing processes are integrated by breadth-first search (BFS) in object parsing with AoG formulated under the Bayesian framework. The three processes are explicitly connected to the Bayesian inference and the dynamic programming (DP) implementation. With experiments on human face parsing and hierarchical image structure parsing, the results show performance improvement in the same manner consistent with their evaluated information contributions.

Next, to advance computational efficiency of learnt computing processes given allowable bounds on accuracy performance in cost-sensitive object detection, near-optimal decision policies are learnt for computing processes of terminal nodes and And-nodes by minimizing the corresponding risk function which explicitly takes into account the computational cost, the false negative (FN) and false positive (FP) loss costs.

Finally, a theoretical study is proposed to schedule all the $\alpha$-$\beta$-$\gamma$ computing processes in an AoG under the best-first heuristic search framework to adapt computing orders of nodes in an AoG to different vision tasks and image datasets.
CHAPTER 1

Introduction

1.1 Motivation and Objectives

Vision can be treated as a continuous computing process dynamically guided by vision tasks, which is dated back to David Marr [Marr, 1982]. Also, “The more you look, the more you see”, as Donald Geman pointed out. Concretely, accuracy performance and computational efficiency are two of the most important issues in vision, and the trade-off between them is guided by vision goals. The objective of this thesis is to present a framework of (i) Identifying BU/TD computing processes in a recursively defined hierarchical AoG; (ii) Evaluating the individual information contributions of the identified BU/TD computing processes; (iii) Integrating BU/TD computing processes in an explicit and robust manner to improve accuracy performance, and (iv) Scheduling BU/TD computing processes (i.e., pursuing a computer order) to advance computational efficiency given allowable bounds on accuracy performance and adapt to different vision tasks.

In the literature of object detection and parsing in computer vision, hierarchical models and contextual information are widely used and shown to improve accuracy performance [Grenander and Miller, 2007; Geman et al., 2002; Riesenhuber and Poggio, 1999; Ullman et al., 2002; Zhu and Mumford, 2006; Todorovic and Ahuja, 2008b; Wu et al., 2010; Sudderth et al., 2008; Felzenszwalb et al., 2010; Fidler et al., 2008; Torralba, 2003; Divvala et al., 2009]. In hierarchical
Figure 1.1: The figure illustrates the motivation and intuitive ideas of identifying the three types of BU/TD computing processes for each node in a hierarchical model in this thesis, termed the $\alpha$-$\beta$-$\gamma$ computing process, using human face detection as an example. There are three cases of human faces appearing in the top image in terms of scale and occlusion conditions, each of which entails a different computing process as illustrated in the middle of the figure, i.e., the $\alpha$(face), $\beta$(face) and $\gamma$(face) processes respectively. Some other corresponding examples are shown in the bottom figure.

models, one can observe that certain nodes, such as the human face, are often interpreted in a top-down fashion in the hierarchy. One does that because it is much more effective to detect the full human face than individual facial components.
In contrast, some other nodes such as junctions and handwritings are more effectively explained in a bottom-up manner in the hierarchy. For example, it is very difficult to detect rectangles directly. Instead, parallel lines or L-junctions can be detected first and then those compatible parallel lines or compatible L-junctions are bound together to form rectangles in terms of some explicit constraints. At the same time, if scale and occlusion conditions are taken into account, one may have to adapt different computing strategies for different cases. Fig.1.1 shows three cases of human faces: the first case is a normal situation in which human faces are at middle resolutions and without occlusions, the second contains human faces at higher resolutions but with occlusions and the third contains human faces at extremely low resolution. Intuitively, these three cases entail a hierarchical model for representing human faces and three different computing processes in inference.

Improving the accuracy performance by using hierarchical models is at the expense of increasing on-line computing costs. In object detection and parsing, the most popular computing strategy is the sliding window method where the computing burden of a hierarchical model grows at least linearly in the number of nodes in the model, and the computing cost of each node grows quadratically in the number of testing image pixels (due to the need of exploring scales and orientations). The explosion of computing costs practically prevent a computer vision system from scaling up to hierarchical models with a large number of nodes (e.g., hundreds and even thousands of nodes). In practice, vision tasks determine the importance of different nodes in a hierarchical model in object detection and parsing by specifying their FN and FP loss costs (or equivalently specifying their desired false positive error rates (FPR) and false negative error rates (FNR)). In most existing work, however, the zero-one loss cost is the most frequently used case. So, the following five questions arise naturally,
(1) What computing processes, BU and TD, can be identified for nodes in a hierarchical model?

(2) How much information does each of them contribute for different nodes?

(3) How should they be integrated to improve accuracy performance?

(4) Which node should be tested first, and then which one is the next, and so on, in order to advance on-line computational efficiency of a hierarchical model given allowable bounds on accuracy performance?

(5) How to adapt the computing order of nodes in a hierarchical model to loss cost-sensitive vision tasks?

The five questions above often refer to the long-standing scheduling problem in vision [Ullman, 1984; Thorpe et al., 1996; Blanchard and Geman, 2005; Zhu and Mumford, 2006], and are to be addressed in this thesis.

1.2 Intuitions of Scheduling

Two games people often play can be used to demonstrate the intuitive ideas of scheduling: the mine sweeper and the “twenty-question” game.

When playing the mine sweeper, one starts with knowing nothing about where the mines are (corresponding to the situation of on-line object detection in an input image that the number of objects appearing in the image, and if had any, the locations, scales and orientations of objects, are all unknown). So, the first checking location is selected randomly (i.e., bottom-up). Furthermore, to complete the sweeping task with less time, one often quickly and randomly picks up a small number of locations to check. The game is over if any chose location turns out to be a mine. Otherwise, the promising locations are sorted in terms
of the information propagated from the surrounding checked locations, i.e., their revealed numbers, which is actually the implicit scheduling made by the player. The queue of promising locations are dynamically updated along the sweeping process. If different time limits were specified, one may sweep mines with different strategies in the sense that the way of selecting the next location to check could be changed accordingly. For example, if there is no time limit, one may sweeps mines by sliding over every possible location.

Similarly, in the “twenty-question” game, if one could ask questions with low computational cost (i.e., minimizing pains) but high information gain (i.e., maximizing gains) in the right order (i.e., scheduling), the remaining uncertainty could be reduced most at each step and then the solution can be approached more quickly. The same observation is illustrated in [Blanchard and Geman, 2005].

1.3 Thesis Outline

The remainder of this thesis is organized as follows.

Chapter 2 first introduces the hierarchical AoG representation which embeds acyclic object detection and parsing grammars, and then defines three types of computing processes, BU/TD, for each node \( A \) in an AoG.

Chapter 3 presents a method of isolating the \( \alpha-\beta-\gamma \) computing processes and then trains the three processes separately under the MLE framework, followed by a numerical study of evaluating their individual information contributions with both computer and human experiments.

Chapter 4 shows the explicit connection between the \( \alpha-\beta-\gamma \) computing processes and the Bayesian inference and dynamic programming implementation in object parsing using AoG formulated under Bayesian framework, and then
Chapter 5 presents a method of learning near-optimal decision policies for the scoring functions which can be factorized based on sequential testing strategy by taking into account the computational cost and the FN and FP loss costs. The experimental results show that the computational efficiency is advanced by learning decision policies for computing processes of terminal nodes and And-nodes in an AoG.

Chapter 6 presents a theoretical study of scheduling the $\alpha$-$\beta$-$\gamma$ computing processes under the best-first heuristic search framework, and a agenda-chart scheduling algorithm.

Chapter 7 concludes this thesis and suggests future research directions.

1.4 Contributions

The main contributions of this thesis are summarized as follows.

(1) It presents a numerical study of the bottom-up and top-down computing processes in a recursively defined hierarchical AoG. To the best of our knowledge, it is the first time this is done in the vision literature.

(2) It trains the identified $\alpha$, $\beta$ and $\gamma$ processes separately to reduce contamination by using an isolation procedure. It evaluates information contributions of the three processes individually in both computer and human experiments.

(3) It proposes a pursuit algorithm for object parsing using AoG formulated under Bayesian framework, which connects the $\alpha$, $\beta$ and $\gamma$ processes to Bayesian inference and DP implementation and then integrates them explicitly by BFS.
for robust inference. The algorithm presents a way to link discriminative learning to the Bayes.

(4) It observes that the effectiveness of the BU/TD (i.e., $\alpha$, $\beta$ and $\gamma$) processes depends on the scale and occlusion conditions. The $\alpha$(face) process is stronger than the $\alpha$ processes of facial components, while $\beta$(junctions) and $\beta$(rectangle) work much better than their $\alpha$ processes.

(5) It presents a method of learning near-optimal decision policy for sequentially factorizable scoring functions by taking into account the computational cost, the FP and FN loss costs explicitly, which advances the computational efficiency by learning to compute faster.
CHAPTER 2

Hierarchical AoG Representation and BU/TD Processes

This chapter first introduces the hierarchical AoG representation [Zhu and Mumford, 2006; Geman et al., 2002] to set up the background and fixes the notations used for defining AoG and the BU/TD computing processes in this thesis. Then, for each And-node $A$ in an AoG, three types of computing processes are defined: the $\alpha(A)$ process detects node $A$ directly based on image features (which could be done by using a bottom-up classifier or a top-down template matching), the $\beta(A)$ process computes node $A$ by binding its child node(s) bottom-up and the $\gamma(A)$ process predicts node $A$ top-down from its parent node(s). The identified three types of computing processes elaborate the intuitive ideas illustrated in Fig. 1.1, and are the main objects of analysis throughout this thesis.

2.1 Hierarchical AoG Representation

An AoG represents a hierarchical model recursively defined for effective visual knowledge representation which embodies a stochastic context sensitive image grammar (SCSG) [Zhu and Mumford, 2006]. The SCSG combines the reconfigurability of stochastic context free grammar (SCFG) with the contextual constraints of graphical Markov random field (MRF) models. Generally, an AoG
Figure 2.1: (Adapted from [Zhu and Mumford, 2006]) The figure illustrates the recursively defined hierarchical And-Or graph (AoG) representation (only a portion is shown for clarity). There are three types of nodes: an And-node representing the decomposition (e.g., an object is a collection of parts), an Or-node representing alternative ways for the decomposition and a Terminal-node used to link the symbol to image data. Note that all And-nodes can directly terminate to image data (through a terminal-node) when it is at low resolution. In traditional hierarchical models, only leaf nodes link to image data.

can represent the structural, geometric, appearance, and probabilistic information for an object category. As illustrated in Fig.2.1, there are three types of nodes in an AoG: And-nodes represent decomposition and are denoted by solid circles, Or-nodes represent alternative structures and are denoted by dash circles and terminal nodes link to image data and are denoted by solid rectangles. In this thesis, each And-node in the AoG can also directly terminate to image data (through a terminal node) when it is at low resolution. In contrast, traditional
hierarchical models do not have Or-nodes and allow only leaf nodes to link to image data [Riesenhuber and Poggio, 1999; Aycinena et al., 2008].

2.2 Definition of an AoG

In this thesis, an AoG is specified by a quadruple,

\[ \mathcal{G} = (V_N, V_T, E, \mathcal{P}) \]  \hspace{1cm} (2.1)

(1) \( V_N = V_{\text{and}} \cup V_{\text{or}} \) is a set of nonterminal nodes consisting of an And-node set \( V_{\text{and}} \) representing decompositions and an Or-node set \( V_{\text{or}} \) representing alternative ways for decompositions. In this thesis, a nonterminal node is denoted by capital letters, e.g., \( A \in V_{\text{and}}, O \in V_{\text{or}} \) in Fig. 2.1.

(2) \( V_T \) is a set of terminal nodes linking And-node symbols to image data, each of which is denoted by a lower-case \( t \) with the subscript letter of the corresponding nonterminal node, e.g., \( t_A \in V_T \).

(3) \( E = E_{\text{or}} \cup E_{\text{dec}} \cup E_t \cup E_{\text{rel}} \) is a set of edges encoding grammar rules for hierarchical structures and contextual relations among symbols and including four types:

(i) \( E_{\text{or}} = \{ < O, A > : O \in V_{\text{or}}, A \in V_{\text{and}} \} \) is a set of vertical switching edges which link Or-nodes to corresponding And-nodes (i.e., alternatives).

(ii) \( E_{\text{dec}} = \{ < A, C > : A \in V_{\text{and}}, C \in ch(A), ch(A) \neq \emptyset \} \) is a set of vertical decomposition edges which connect And-nodes to their child nodes (And-nodes or Or-nodes). \( ch(A) \subset V_N \) denotes the set of child nodes of node \( A \).

(iii) \( E_t = \{ < A, t_A > : A \in V_{\text{and}}, t_A \in V_T \} \) is a set of vertical terminating edges which connect And-nodes to their corresponding terminal nodes.
(iv) \( E_{rel} = \{ < B, C > : B, C \in V_{\text{or}}, \text{prt}(B) \cap \text{prt}(C) \neq \emptyset \} \) is a set of horizontal relation edges which connect among And-nodes at the same layer, often pairwise. \( \text{prt}(B), \text{prt}(C) \subset V_N \) denotes the sets of parent nodes of nodes \( B \) and \( C \) respectively.

(4) \( \mathcal{P} \) is the probability defined over the space of all valid parse graphs which are defined in Sec. 2.2.1.

Table 2.1 summarizes the notations used for defining AoG and the BU/TD computing processes in this chapter.

Attributes. In an AoG \( \mathcal{G} \), each Or-node \( O \in V_{\text{or}} \) has a switching variable indicating the occurring frequency of its branches, denoted by \( p(A|O) \) \( (A \in V_N) \). Both And-nodes \( A \in V_{\text{and}} \) and terminal nodes \( t \in V_T \) have a vector of attributes denoted by \( X(A) \) and \( x(t) \) respectively. For a subset \( v \subset V_{\text{and}} \), denote by \( X(v) \) the concatenation of attributes for And-nodes in the subset \( v \). The attributes often include locations, scales, orientations, etc.. The attributes of an And-node can be passed from attributes of other nodes in three ways: (i) the corresponding terminal node directly, (ii) child And-node(s) during binding process or (iii) parent node(s) during prediction process.

2.2.1 Parse graph

As illustrated by the dark arrows in the left figure of Fig.2.1, a parse graph, \( pg \), is an instance of an AoG constructed by selecting variables at the Or-nodes and specifying the attributes for And-nodes through the three ways stated above, and is defined by,

\[
pg = (V^pg_N, V^pg_T, E^pg, p(pg))
\]  

(2.2)
where $V_{N}^{pg} = V_{N}^{pg \text{ and}} \cup V_{or}^{pg}$ ($V_{N}^{pg} \subset V_{N}$) is the nonterminal node set of the parse graph, $V_{T}^{pg} \subset V_{T}$ is the terminal node set and $E^{pg} = E^{pg \text{ or}} \cup E^{pg \text{ dec}} \cup E^{pg \text{ t}} \cup E^{pg \text{ rel}}$ ($E^{pg} \subset E$) is the edge set. a parse graph becomes a parse tree if the horizontal relation edges $E^{pg \text{ rel}} \subset E^{pg}$ is omitted.

$p(pg)$ is the prior probability of parse graph $pg$, measuring the occurring probability of each switching edge $< O, A > \in E^{pg \text{ or}}$ and the compatibility probabilities among the attributes of And-nodes (pairwise used in this paper) in $V^{pg \text{ and}}$ with vertical decomposition edge $< P, A > \in E^{pg \text{ dec}}$ and horizontal relation edge $< C_i, C_j > \in E^{pg \text{ rel}}$. So, $p(pg)$ is defined by,

$$p(pg) = \frac{1}{Z} \exp\{-\mathcal{E}(pg)\} \quad (2.3)$$

where $Z = \sum_{pg} \exp\{-\mathcal{E}(pg)\}$ is the partition function and $\mathcal{E}(pg)$ is the total energy defined by,

$$\mathcal{E}(pg) = - \sum_{<O,A> \in E^{pg \text{ or}}} \log p(A|O)$$

$$- \sum_{<P,A> \in E^{pg \text{ dec}}} \log p(X(A)|X(P))$$

$$- \sum_{<C_i,C_j> \in E^{pg \text{ rel}}} \log p(X(C_i),X(C_j)) \quad (2.4)$$

and $p(A|O)$ is the switching probability estimated by the occurring frequency in training data [Zhu and Mumford, 2006], $p(X(A)|X(P))$ captures the top-down prediction model and $p(X(C_i),X(C_j))$ captures the compatibilities in the bottom-up binding model. They will be specified in the learning of the $\alpha$-$\beta$-$\gamma$ computing processes.

Given an input image $I$ with domain defined on lattice $\Lambda$, the inference of AoG is to construct a parse graph for each object instance and its structure is not predefined but inferred on the fly.
2.2.2 Configuration

As illustrated in the right figure of Fig. 2.1, a configuration, \( C \), is the set of all terminal nodes in a valid parse graph \( pg \), flattened in an image lattice.

\[
C(pg) = \{(t, x(t)) : t \in V_{T}^{pg}\}
\] (2.5)

The image data likelihood of a parse graph \( pg \), \( p(I|pg) \), is measured based on the terminal nodes in \( V_{T}^{pg} \) (since they link to image data). Further, if there was no occlusion between different terminal nodes (which is true for roughly rigid object categories such as the human face), the likelihood is factorized as,

\[
p(I|pg) = p(I|C(pg)) = \prod_{t \in V_{T}^{pg}} p(I_{\Lambda_{t}}|t)
\] (2.6)

where \( \Lambda_{t} \in \Lambda \) is the image domain occupied by the terminal node \( t \).

In inference, \( p(I|pg) \) does not need to be computed exactly, instead the likelihood ratio between \( p(I|pg) \) and a reference background model is measured and the reference background model is made implicitly in the later derivation in this thesis.

2.3 The BU/TD Computing Processes Identified in an AoG

2.3.1 Background

In hierarchical models, bottom-up and top-down are two basic computing mechanisms and are often used with three strategies in the literature:

(1) Pure bottom-up inference which passes messages in a feed-forward manner in the hierarchy, starting from data-driven features [Riesenhuber and Poggio, 1999; Serre et al., 2007; Aycinena et al., 2008].
(2) Pure top-down inference which passes messages in a feed-back manner in the hierarchy, starting from template matching [Todorovic and Ahuja, 2008a; Demirci et al., 2009].

(3) One pass of bottom-up inference followed by one phase of top-down inference [Tu et al., 2005; Epshtein et al., 2008; Borenstein and Ullman, 2008; Levin and Weiss, 2009; Demirci et al., 2006].

In the recent vision literature, it is well acknowledged that both bottom-up and top-down inference processes contribute to object detection, recognition and parsing, and they should be combined [Lee and Mumford, 2003; Jin and Geman, 2006]. Despite many efforts, it has been unclear how to combine bottom-up and top-down inference processes in a robust and effective way. The first numerical evaluation of top-down versus bottom-up is the ROC comparisons addressed in [Han and Zhu, 2009]. Another work on compositional boosting [Wu et al., 2007] proposed to separate the implicit testing (i.e., the $\alpha$ process to be defined below) and explicit testing (i.e., the $\beta$ process to be defined below) and then combine them under the compositional boosting. This thesis presents a more general framework and formulation to integrate the bottom-up and top-down inference processes (i.e., the $\alpha$, $\beta$ and $\gamma$ processes to be defined below) in an explicit way so that different kinds of integrations can be compared numerically.

### 2.3.2 Definitions of the $\alpha$-$\beta$-$\gamma$ Processes

Fig.2.2 shows a portion of an AoG using the face example discussed in Fig.1.1 where node $A$ represents human face, node $P$ represents head-shoulder and node $C_i$’s represent facial components ($i = 1, 2, 3$). As an AoG is recursively defined, the $\alpha$, $\beta$ and $\gamma$ processes are presented by considering an And-node $A$ as illus-
Figure 2.2: Illustration of identifying the $\alpha(A)$, $\beta(A)$ and $\gamma(A)$ inference processes for each And-node $A$ in an AoG. The $\alpha(A)$ process is directly based on the compact image data of node $A$ (either bottom-up or top-down), the $\beta(A)$ process generates hypotheses of node $A$ by bottom-up binding the $\alpha$ processes of some child node(s) (for example, $\{\alpha(C_1), \alpha(C_2)\} \rightarrow \beta(A)$), and the $\gamma(A)$ process predicts hypotheses of node $A$ from the $\alpha$ processes of some parent node(s) (for example, $\alpha(P) \rightarrow \gamma(A)$ or $\beta(A) \rightarrow \gamma(C_3)$ in a top-down fashion). In computing, each process has two states: “on” or “off”, for example, $\alpha(C_3)$ process is off and it is plotted in grey. As an AoG is defined recursively, each And-node has its own $\alpha$, $\beta$ and $\gamma$ processes (except that the root node’s $\gamma$ processes and the $\beta$-processes of leaf nodes are always off).

As demonstrated in Fig.2.2 without loss of generality, and defined as follows [Wu and Zhu, 2011].
Definition 1: (the $\alpha$ process). The $\alpha(A)$ process handles situations in which node $A$ is at middle resolution without occlusion. Node $A$ can be detected directly (based on its compact image data) and alone (without taking advantage of surrounding context) while its children or parts are not recognizable alone in cropped patches. An example of $\alpha$ (face) process is shown in the left-bottom panel of Fig.1.1. Most of the sliding window detection methods in computer vision literature belong to this process. It can be viewed as either bottom-up or top-down. By bottom-up, it means that discriminative models are used to train the $\alpha$ process, such as the Adaboost classifiers [Viola and Jones, 2004]. By top-down, it means that generative models are used, such as the active basis model [Wu et al., 2010].

Definition 2: (the $\beta$ process). When node $A$ is at high resolution, it is more likely to be occluded in a scene. Node $A$ itself is not detectable in terms of the $\alpha(A)$ process due to occlusion. A subset of node $A$’s child nodes can be detected in cropped patches (say, their $\alpha$ processes are activated). Then, the $\beta(A)$ process computes node $A$ by binding the detected child nodes bottom-up under some compatibility constraints. An example of $\beta$ (face) process is illustrated in the middle-bottom panel of Fig.1.1. Most of component [Biederman, 1987; Heisele et al., 2007], fragment [Ullman et al., 2002] or part [Amit and Trouvé, 2007; Schneiderman and Kanade, 2002] based methods, the constellation models [Fei-Fei et al., 2006; Fergus et al., 2007] and the pictorial models [Felzenszwalb and Huttenlocher, 2005] belong to this process.

Definition 3: (the $\gamma$ process). The $\gamma(A)$ process handles situations in which node $A$ is at very low resolution. Node $A$ can not be detected alone in isolation based on $\alpha(A)$, and neither can its parts. Then, the $\beta(A)$ process also fails. An
Figure 2.3: The figure illustrates how the $\alpha$-$\beta$-$\gamma$ processes communicate each other.

An example of $\gamma$(face) process is illustrated in the right-bottom panel of Fig. 1.1. So, information outside of the local window must be incorporated. The $\gamma(A)$ process predicts node $A$ top-down from a parent node whose $\alpha$ process is activated. In this thesis, contextual information are passed on by the parent node, such as information from some sibling nodes or other spatial context. Most of the context-based methods [Torralba, 2003; Hoiem et al., 2008; Fink and Perona, 2003] belong to this process.

For an And-node $A$, all the three inference processes, $\alpha(A)$, $\beta(A)$ and $\gamma(A)$, contribute to computing it from images in complementary ways (see Fig. 2.3). The $\alpha(A)$ process “sees” node $A$ directly based on the image feature without “thinking” any concepts about parts or contextual information of node $A$ and it is often based on a log-likelihood ration test. The $\beta(A)$ process “sees” node $A$ by “thinking” the compatibility after “seeing” the parts in a given step, and similar for the $\gamma$ process. The effectiveness of each process depends on the scale
Figure 2.4: Illustration of integrating the $\alpha(\text{face})$, $\beta(\text{face})$ and $\gamma(\text{face})$ in the human face AoG for face detection. The three inference processes are effective in complementary ways relatively depending on the scale and occlusion conditions. The typical situations shown here are common to other object categories.

and occlusion conditions. As shown in Fig.2.4, the three cases of human faces shown in Fig.1.1 can be handled by the $\alpha(\text{face})$, $\beta(\text{face})$ and $\gamma(\text{face})$ respectively. Intuitively, for robust inference one should integrate them. As an AoG is a recursive structure, the three inference processes are also defined recursively and each And-node has its own $\alpha$, $\beta$ and $\gamma$ inference processes (except that the $\gamma$ process of the root node and the $\beta$ processes of leaf nodes are always disabled).
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Λ</td>
<td>image lattice</td>
</tr>
<tr>
<td>(I_Λ)</td>
<td>an image (I) defined on the lattice (Λ)</td>
</tr>
<tr>
<td>(G = (V_N, V_T, E, P))</td>
<td>a quadruple representing the AoG (G)</td>
</tr>
<tr>
<td>(V_N, V_{and}, V_{or})</td>
<td>a nonterminal node set (V_N) consisting of an And-node set (V_{and}) and an Or-node set (V_{or}).</td>
</tr>
<tr>
<td>(V_T)</td>
<td>a terminal node set with one-to-one mappings to And-nodes</td>
</tr>
<tr>
<td>(E, E_{or}, E_{dec}, E_t, E_{rel})</td>
<td>a set (E) of edges consisting of a switching edge set (E_{or}), a decomposition edge set (E_{dec}), a terminal edge set (E_t) and a relation edge set (E_{rel})</td>
</tr>
<tr>
<td>(V_{and}^{ch}, V_{and}^{prt} \subset V_{and})</td>
<td>subsets of And-nodes which have child node(s) and parent node(s) respectively</td>
</tr>
<tr>
<td>(ch(), \text{pr}t())</td>
<td>child node set and parent node set such as (ch(A), \text{pr}t(A) \subset V_N)</td>
</tr>
<tr>
<td>(X(), x())</td>
<td>a vector of attributes of a And-node such as (X(A)) or a terminal node such as (x(t_A)) respectively, often including the relative locations, scales and orientations</td>
</tr>
<tr>
<td>(pg)</td>
<td>a parse graph which is an instance of the AoG (G)</td>
</tr>
<tr>
<td>(C(pg))</td>
<td>a configuration collapsed from (pg) to image lattice (Λ)</td>
</tr>
<tr>
<td>(α(A; θ))</td>
<td>(α) computing process of an And-node (A)</td>
</tr>
<tr>
<td>(β(A</td>
<td>c; φ))</td>
</tr>
<tr>
<td>(γ(A</td>
<td>P; ϕ))</td>
</tr>
</tbody>
</table>

Table 2.1: The table of notations for defining AoG and \(α-β-γ\) processes
CHAPTER 3

Evaluating Information Contributions of 
BU/TD Processes

This chapter presents a method of evaluating the individual information contributions of the $\alpha$-$\beta$-$\gamma$ processes for computing a node $A$ in an AoG. To that end, the $\alpha(A)$, $\beta(A)$ and $\gamma(A)$ processes are isolated by blocking the other two processes and then trained separately. The training data are generated based on labelled parse graphs by using the proposed isolation method. The learning of the three processes are formulated under the MLE framework. The $\alpha$ process adopts the active basis model [Wu et al., 2010], and the $\beta$ and $\gamma$ processes use the pictorial structure [Fischler and Elschlager, 1973; Felzenszwalb and Huttenlocher, 2005]. Then, the information contribution of each process is evaluated individually based on the discriminative power, compared with the human performance obtained by a human study. With experiments on both hierarchical image structures, such as L-junctions and rectangles, in low-to-middle-level vision, and objects, such as human faces, in high-level vision, the results show that (i) All the three processes contribute to computing node $A$ from images in complementary ways; (ii) The effectiveness of the $\alpha(A)$, $\beta(A)$ and $\gamma(A)$ processes depends on the scale and occlusion conditions as motivated in Fig.1.1; and (iii) The $\alpha$(face) process is stronger than the $\alpha$ processes of facial components, while $\beta$(junctions) and $\beta$(rectangle) work much better than their respective $\alpha$- processes.
3.1 Isolating BU/TD Computing Processes

3.1.1 Motivation

In this chapter, the $\alpha$-$\beta$-$\gamma$ processes are trained separately based on their respective isolated training data. The motivation and necessity to do that are as follows. When learning a human face classifier (i.e., the $\alpha$(face) process), one may select positive examples with two different ways: (i) Only face examples like those pointed by the $\alpha$(face) arrow in Fig.2.4, or (ii) A set of human face examples mixing all those shown in the right middle box in Fig.2.4. In practice, positive examples are obtained by cropping image patches often only based on the labelled bounding boxes. When labelling the bounding box for an object instance, however, one might already take advantage of all the information coming from the $\alpha$, $\beta$ and $\gamma$ processes. Actually, the $\alpha$ processes in most existing work is trained based on a set of mixed positive examples (especially, mixing the $\alpha$ case and the $\gamma$ case). Then, the learned classifier could be contaminated depending on the mixing rate implicitly [Torralba et al., 2007; Fink and Perona, 2003; Avidan, 2006]. Whether a feature is selected to be used by a classifier depends on how different the feature responses of positive examples and those of negative examples are. The feature responses computed from a set of mixed positive examples do not reflect the true discriminative power of the feature.

3.1.2 The Isolating Method

Since scale and occlusion are the two main causes entailing the $\alpha$, $\beta$ and $\gamma$ processes, each of the three computing processes of a node $A$ can be blocked through scaling and/or masking image patches of node $A$ in terms of the labelled parse graphs. Then, one process is isolated by blocking the other two processes. Fig.3.1
Figure 3.1: Illustration of isolating the $\alpha$, $\beta$ and $\gamma$ processes of a node $A$ in an AoG based on the labelled parse graphs. Consider the human face example in this figure. The isolation is performed by scaling and masking the image patch (the left-top panel) in terms of its annotation (the left-bottom panel). The annotation used in this chapter is the manually labelled parse graph. Based on the isolation, training and evaluation data are generated for each process.

illustrates the procedures with an example for human face (node $A$).

**Isolating the $\alpha(A)$ process.** Block both the $\beta(A)$ and $\gamma(A)$ processes will isolate the $\alpha$ process. First, only the compact image patches of node $A$ are cropped out of its context in terms of the annotations. Then, the image patches are down-sampled up to a certain scale at which the parts can not be recognized if cropped in isolation.

**Isolating the $\beta(A)$ process.** There are different $\beta(A)$ processes depending on the given set of node $A$’s child nodes, $c \subseteq ch(A)$. Then, to isolate the $\beta(A|c)$ process is is to block both the $\alpha(A)$ and $\gamma(A)$ processes meanwhile keep the $\alpha$ processes of child nodes in $c$ on. First, only the compact image patches of node $A$ are cropped out but just keep those patches whose resolutions are above a predefined value. Then, all the image patches are resized to the same size (also
above the predefined value) and the portions of all the image patches are masked with respect to the child node(s) not in c.

**Isolating the $\gamma(A)$ process.** Similarly, there might have different $\gamma$ processes for node $A$ given different parent node $P$. To isolate the $\gamma(A|P)$ process is to block the $\alpha(A)$ and $\beta(A)$ while keeping the $\alpha(P)$ process on. So, it is equivalent to isolate the $\alpha(P)$ process. First, the compact image patches of the parent node $P$ are cropped out, and then are down-sampled to a certain scale at which the node $A$ itself can not be recognized if cropped in isolation.

If a testing image dataset was manipulated deliberately by these isolating methods, a single computing process will fail to obtain good accuracy performance without doubt. So, to achieve robust performance, the three computing processes are trained separately and then are integrated explicitly in this thesis.

Next, the method of generating training and evaluation data for each process in terms of the isolating procedures is presented.

### 3.2 Generating Training Data for BU/TD Computing Processes

Suppose a set of $m$ positive images for an object of interest is denoted by,

$$D^+ = \{(I_1, pg_1), \ldots, (I_m, pg_m)\} \quad (3.1)$$

where $pg_i$ is the annotated parse graph for image $I_i$. Based on the parse graph, the training and testing datasets for the three processes are generated by using the isolating methods. For simplicity of notations, assume that each node of interest appears in each image $I_i$ with good resolution.
The α process training dataset. Let $D^+_\alpha(A)$ denote the positive training dataset for the $\alpha(A)$ process of node $A$. By using the isolation method of the $\alpha$ process, for each $I_i \in D^+$, the $\alpha$ image patch of node $A$ is obtained, denoted by $I_i^{(A)}$, and then the $\alpha(A)$ process training dataset is denoted by,

$$D^+_\alpha(A) = \{I_i^{(A)} : i = 1, 2, \cdots, m\} \tag{3.2}$$

The β process training dataset. Let $D^+_\beta(A|c)$ denote the positive training dataset for the $\beta(A|c)$ process. Through the isolation method of the $\beta$ process, for each $I_i \in D^+$, the $\beta$ image patch of node $A$ given child node(s) in $c$ is obtained, denoted by $I_i^{(c)}$, and then the $\beta(A|c)$ process training dataset is denoted by,

$$D^+_\beta(A|c) = \{(I_i^{(c)}, X(c|I_i^{(c)})) : i = 1, 2, \cdots, m\} \tag{3.3}$$

where $X(c|I_i^{(c)}) = \{X(C_j) : C_j \in c, j = 1, \cdots, |c|\}$ is the concatenation of attributes for child node(s) in $c$ measured in $I_i^{(c)}$, which will be used to learn the bottom-up $\beta$ binding model $p(X(C_j), X(C_k))’s$ for node $A$ given child nodes in $c$ ($j \neq k$).

The γ process training dataset. Let $D^+_\gamma(A|P)$ denote the positive training dataset for the $\gamma(A|P)$ process. By adopting the isolation method of the $\gamma$ process, for each $I_i \in D^+$, the $\gamma$ image patch of node $A$ given the parent node $P$ is generated and denoted by $I_i^{(P)}$, and then the $\gamma(A|P)$ process training dataset is denoted by,

$$D^+_\gamma(A|P) = \{(I_i^{(P)}, X(A|I_i^{(P)})) : i = 1, 2, \cdots, m\} \tag{3.4}$$

where $X(A|I_i^{(P)})$ is the attributes of node $A$ measured in $I_i^{(P)}$, which will be used to learn the top-down $\gamma$ prediction model $p(X(A)|X(P))$ of node $A$ given the parent node $P$. When node $P$ only has one child node $A$, the $\gamma$ prediction model
of node $A$ given the parent node $P$ (i.e., $p(X(A)|X(P))$) could be transformed into the $\beta$ binding model of node $P$ given its child node $A$ equivalently.

In the same way, the attributes $X(C_j|I_i^{(A)})$’s can be calculated to learn top-down $\gamma$ prediction models $p(X(C_j)|X(A))$’s for child nodes $C_j$’s of node $A$ where $C_j \in ch(A)$, $I_i^{(A)} \in D^+_\alpha(A)$.

Correspondingly, the negative datasets $D^-_\alpha(A)$, $D^-_\beta(A|c)$ and $D^-_\gamma(A|P)$ consist of two sets: one is “easy” negative dataset which are generated by randomly cropping image patches from background images which do not contain the objects, and the other the “hard” negative dataset which are generated by a data-mining strategy, i.e., running the computing processes in the background images to collect false positive samples.

**Scale specifications in experiments.** In the experiments for evaluating the information contributions, the training and evaluation data are generated with multiple scales used to investigate how the information contributions change with scales. In the experiments for object parsing by integrating the $\alpha$, $\beta$ and $\gamma$ processes of node $A$ in Chapter 4, fixed relative scales are used for the three processes. Denote by $s_{\alpha(A)}$, $s_{\beta(A)}$ and $s_{\gamma(A)}$ the scales of the compact image patches of node $A$ in the three processes respectively, and the fixed relative scales mean $s_{\alpha(A)} = b \times s_{\gamma(A)} = \frac{1}{b} \times s_{\beta(A)}$ (e.g., $b = 2$).

Given the training and testing data generated, the learning of the three processes are formulated under the MLE framework in the next section.
3.3 Learning BU/TD Computing Processes by MLE

3.3.1 Learning the $\alpha$ Process

Learning the $\alpha$ process involves selecting a modeling scheme for $\alpha(A; \theta)$ and estimating the parameters $\theta$ by maximizing the data likelihood on $D_\alpha^+(A)$. For example, in discriminative boosting methods, $\theta_A$ is the learned strong classifier which consists of a set of boosted weak classifiers and the corresponding weights [Viola and Jones, 2004], and in generative model-based methods such as the active basis model, $\theta$ is the set of parameters specify the learned deformable template [Wu et al., 2010]. Given $D_\alpha^+(A)$, the MLE learning of $\alpha$ process is defined by,

$$\alpha(A; \theta^*) = \arg \max_\theta p(D_\alpha^+(A)|A; \theta)$$

$$= \arg \max_\theta \sum_{i=1}^m \log p(I_i(A)|A; \theta) \quad (3.5)$$

Solving $\alpha(A; \theta^*)$ depends on choosing a specific modeling scheme for $p(I|A; \theta)$. In this chapter, the active basis model is used and briefly introduced here for this chapter to be self-contained.

**Active Basis Model.** The active basis model is a deformable model which consists of a small number of Gabor wavelet elements (as visual primitives for modeling object category) at selected locations and orientations. These Gabor wavelet elements can slightly perturb their locations and orientations before they are linearly combined to generate the observed image. Let $\Lambda$ be the domain of the image patch $I$ and $\{B_{x,y,s,o}\}$ the dictionary of Gabor wavelet elements. The $(x, y, s, o)$ are densely sampled: $(x, y) \in \Lambda$, $s$ is a fixed size (often about 1/10 of the length of $\Lambda$) and $o \in \{i\pi/N, i = 0, \ldots, N - 1\}$ (e.g., $N = 15$). The dictionary forms an over-complete dictionary for modeling $I_\Lambda$. Then, the
Figure 3.2: Illustration of learned $\alpha$, $\beta$ binding and $\gamma$ prediction models for human face. The top panel shows the learned active basis model for the $\alpha$ process of each terminal node. The left-bottom panel illustrates the binding model for the $\beta$ process in which the outside red box is the bounding box of face and the inside dash boxes are for the parts and the ellipses represent the location following a Gaussian distribution. The right-bottom panel shows the prediction model for the $\gamma$ process in which the outside green box is the bounding box of head-shoulder and the inside solid and dash boxes represent the changeable size of the bounding box of face and the red ellipse represent the location of face following a Gaussian distribution.
Figure 3.3: Illustration of the AoG for junctions and rectangles. The left panel shows some positive examples of L junction, cross junction, parallel line, T/Y/Arrow junction. Each sample of L, cross and T/Y/Arrow junctions is shown under three different scales (10 × 10 pixels, 20 × 20 pixels and 30 × 30 pixels) from which it is intuitively observed that the $\alpha$ process would be very weak. The right-top panel shows some samples of rectangle in which it is observed that the $\alpha$ process would not work well due to the variabilities. The right-bottom panel shows the AoG for rectangle.

sparse coding scheme $I = \sum_{i=1}^{n} a_i B_i + U$ is adopted where $n$ is the number of selected bases, $B_i = B_{x_i,y_i,z_i}^i$, $a_i$’s are the coefficients and $U$ is the unexplained residual image. In the matrix form, $I = B a + U$ (where $B = (B_1, \ldots, B_n)$ and $a = (a_1, \ldots, a_n)^T$). In terms of linear decomposition, the residual component $U$ resides in the remaining subspace orthogonal to $B$ and then it is written as $U = \bar{B}\bar{a}$ (where columns in $\bar{B}$ are orthogonal to columns in $B$ and both $\bar{B}$ and $\bar{a}$ would be made implicit in the active basis model). So, the sparse image
representation is \( I = B\mathbf{a} + B\bar{\mathbf{a}} \). Then, the distribution of \( I \) given \( B \) is specified by,

\[
p(I|B) = p(\mathbf{a}, \bar{\mathbf{a}} | B) \det(J) = p(\mathbf{a})p(\bar{\mathbf{a}} | \mathbf{a}) \det(J)
\]

(3.6)

where \( J \) is the Jacobi matrix of the linear transform from \( I \) to \((\mathbf{a}, \bar{\mathbf{a}})\) and \( \det(J) \) the determinant of \( J \).

On the other hand, let \( q(I) \) be a reference distribution (which has a few choices discussed in [Wu et al., 2010]), and similarly, it is rewritten as,

\[
q(I) = q(\mathbf{a}, \bar{\mathbf{a}} | B) \det(J) = q(\mathbf{a})q(\bar{\mathbf{a}} | B) \det(J)
\]

(3.7)

In the active basis model, the objective is to construct \( p(I|B) \) by modifying \( q(I) \) under the assumption \( q(\bar{\mathbf{a}} | \mathbf{a}) = p(\bar{\mathbf{a}} | \mathbf{a}) \),

\[
p(I|B) = q(I) \prod_{i=1}^{n} \frac{p(a_i)}{q(a_i)} = q(I) \prod_{i=1}^{n} \frac{p(a_i, \cdots, a_n)}{q(a_1, \cdots, a_n)}
\]

(3.8)

Furthermore, by applying the local inhibition principle, the selected Gabor wavelet elements compete each other to explain away their corresponding image domain and then can be pursued independently, that is,

\[
p(I|B) = q(I) \prod_{i=1}^{n} \frac{p(a_i)}{q(a_i)}
\]

(3.9)

where \( p(a_i) \) is parametrized as an exponential family model

\[
p(a_i; \lambda_i) = \frac{1}{Z(\lambda_i)} \exp\{\lambda_i h(r_i)\}q(a_i)
\]

and \( r_i = | < I, B_i > |^2 \) is the local energy of Gabor filter response and \( h(r_i) = Sigmoid(r_i) = \zeta \left[ \frac{2}{1+e^{-2r_i/\zeta}} - 1 \right] \) is a sigmoid transformation function with \( \zeta \) being the saturation level such as \( \zeta = 6 \). \( q(a_i) \) is pooled from generic background images at an off-line stage. The resulting model is

\[
p(I|B) = q(I) \prod_{i=1}^{n} \frac{1}{Z(\lambda_i)} \exp\{\lambda_i h(r_i)\}
\]

(3.10)
In testing, the matching score (the $\alpha$ weight of a hypothesis of node $A$) is the log-likelihood ratio,

$$w_A^\alpha = \log \frac{p(I|B)}{q(I)} = \sum_{i=1}^{n} \left[ \lambda_i h(r_i) - \log Z(\lambda_i) \right] \quad (3.11)$$

Active basis model can be also used to learn mixed image template modeling both shape and texture [Wu et al., 2010; Si et al., 2009]. The top panel of Fig.3.2 shows the learned active basis for each terminal node of a face AoG.

The threshold $\text{Th}(w_A^\alpha)$ of the $\alpha$ process $\alpha(A; \theta)$ can be estimated in a validation $\alpha$ dataset.

### 3.3.2 Learning the $\beta$ Process

Learning the $\beta$ process involves specifying $\beta(A|c; \phi)$ and estimating the parameters $\phi$ by maximizing the data likelihood on $D_\beta^+(A|c)$. $\beta(A|c; \phi)$ composes (or binds) a (complete or partial) set of child nodes in $c$, to generate hypotheses of node $A$. In the literature, component-based [Biederman, 1987; Heisele et al., 2007], fragment-based [Ullman et al., 2002] and other part-based methods [Amit and Trouvé, 2007; Wu et al., 2007] can be treated as this kind of process. Given $D_\beta^+(A|c)$, the MLE learning of $\beta$ process is defined by,

$$\beta(A|c; \phi^*) = \arg\max_{\phi} p(D_\beta^+(A|c)|c; \phi)$$

$$= \arg\max_{\phi} \sum_{i=1}^{m} \log p(I_i^{(e)}, X(c|I_i^{(e)}|c; \phi)) \quad (3.12)$$

The $\beta(A|c; \phi)$ process includes two components, one is the $\alpha$ process of each child node in $c$ with parameters $\theta_c$ and the other is the binding model for the given children $c$ with parameters $\triangle$. So, $\phi = (\theta_c, \triangle)$ and,

$$p(I^{(e)}, X(c|I^{(e)}|c; \phi) = p(I^{(e)}|c; \theta_c) \times p(X(c|I^{(e)}); \triangle) \quad (3.13)$$
where for notational simplicity $I^{(c)}$ is used to represent $I_i^{(c)}$.

In this chapter, three types of attributes are considered for binding, the location $L_c$, scale $S_c$ and orientation $O_c$ respectively. So, $\Delta = (\Delta_L, \Delta_S, \Delta_O)$. And, they are modeled with pairwise relation used. Consider $c = (C_i, C_j)$ ($C_i, C_j \in ch(A)$), the attributes are,

$$X(c|I^{(c)}) = \{X(C_i), X(C_j)\} = (L_c, S_c, O_c|I^{(c)})$$

and

$$L_c = (L_{C_i}, L_{C_j}); \ S_c = (S_{C_i}, S_{C_j}); \ O_c = (O_{C_i}, O_{C_j})$$

The $\alpha$ processes of child nodes in $c$ are specified by,

$$\log p(I^{(c)}|c) = \log p(I^{(C_i,C_j)}|C_i, C_j) \tag{3.14}$$

$$= \log p(I^{(C_i)}|C_i; \theta_{C_i}) + \log p(I^{(C_j)}|C_j; \theta_{C_j})$$

$$= \alpha(C_i; \theta_{C_i}) + \alpha(C_j; \theta_{C_j})$$

For binding child nodes in $c$, the model of attributes are specified by,

$$p(X(c|I^{(c)}); \Delta) = P(X(C_i), X(C_j); \Delta) \tag{3.15}$$

$$= p(L_c, S_c, O_c|I^{(c)}; \Delta)$$

$$= p(L_c|I^{(c)}; \Delta_L) \times p(S_c|I^{(c)}; \Delta_S) \times p(O_c|I^{(c)}; \Delta_O)$$

Solving $\beta(A|c; \phi^*)$ depends on choosing a specific modeling scheme for the three probability distributions $p(L_c|I^{(c)}; \Delta_L)$, $p(S_c|I^{(c)}; \Delta_S)$ and $p(O_c|I^{(c)}; \Delta_O)$.

In this chapter, the three terms are modeled as Gaussian distributions in their respective transformed spaces [Felzenszwalb and Huttenlocher, 2005]. Thus, each pairwise binding $c = (C_i, C_j)$ is characterized by the expected relative location $\mu_{L_c}$, scale $\mu_{S_c}$ and orientation $\mu_{O_c}$ and corresponding full covariance matrices
\[ \Sigma_{L_c}, \Sigma_{S_c} \text{ and } \Sigma_{O_c}. \] So, \( \Delta_L = (\mu_{L_c}, \Sigma_{L_c}) \), \( \Delta_S = (\mu_{S_c}, \Sigma_{S_c}) \) and \( \Delta_O = (\mu_{O_c}, \Sigma_{O_c}) \) which can be estimated from the dataset \( D_\beta^+(A|c) \), and then their models are expressed as,

\[
p(L_c|I^{(c)}; \Delta_L) = p(L_{C_i}, L_{C_j}; \mu_{L_c}, \Sigma_{L_c}) = \mathcal{N}(L_{C_i} - L_{C_j}; \mu_{L_c}, \Sigma_{L_c}) \tag{3.16}
\]

\[
p(S_c|I^{(c)}; \Delta_S) = p(S_{C_i}, L_{C_j}; \mu_{S_c}, \Sigma_{S_c}) = \mathcal{N}(S_{C_i} - S_{C_j}; \mu_{S_c}, \Sigma_{S_c}) \tag{3.17}
\]

and

\[
p(O_c|I^{(c)}; \Delta_O) = p(O_{C_i}, O_{C_j}; \mu_{O_c}, \Sigma_{O_c}) = \mathcal{N}(O_{C_i} - O_{C_j}; \mu_{O_c}, \Sigma_{O_c}) \tag{3.18}
\]

Furthermore, the three Gaussian distributions above are specified in a transformed space to have zero means and diagonal covariances [Felzenszwalb and Huttenlocher, 2005]. To that end, the singular value decompositions (SVD) of the three covariance matrices is computed and then the transformations are defined. For example, for the location (the same is for the scale and orientation), the transformations are,

\[
\Sigma_{L_c} = U_{L_c} D_{L_c} U_{L_c}^T \tag{3.19}
\]

\[
T_{ij}(L_{C_i}) = U_{L_c}^T (L_{C_i} - \mu_{L_c}) \tag{3.20}
\]

\[
T_{ji}(L_{C_j}) = U_{L_c}^T (L_{C_j}) \tag{3.21}
\]

So, Eqn.3.16 is rewritten as

\[
p(L_c|I^{(c)}; \Delta_L) = \mathcal{N}(L_{C_i} - L_{C_j}; \mu_{L_c}, \Sigma_{L_c}) = \mathcal{N}(T_{ij}(L_{C_i}) - T_{ji}(L_{C_j}); 0, D_{L_c}) \tag{3.22}
\]
Now, the binding score of nodes in \( c \) is calculated by,

\[
w^\text{bind}_c = \log p(X(c|I^{(c)}); \Delta) \tag{3.23}
\]

Then, by combining Eqn.3.14 and Eqn.3.23, the weight of a \( \beta \) hypothesis of node \( A \) given the children \( c \) is defined by,

\[
w^\beta_A(c) = \log p(I^{(c)}, X(c|I^{(c)}); \varphi) \\
= \log p(I^{(c)}|c) + \log p(X(c|I^{(c)}); \Delta) \\
= w^\text{bind}_c + \sum_{C_i \in c} w^\alpha_{C_i} \tag{3.24}
\]

The threshold \( \text{Th}(w^\beta_A(c)) \) of the \( \beta \) process \( \beta(A|c; \varphi) \) can be estimated in a validation \( \beta \) dataset.

### 3.3.3 Learning the \( \gamma \) Process

Learning the \( \gamma \) process involves specifying \( \gamma(A|P, \varphi) \) and estimating the parameters \( \varphi \) by maximizing the data likelihood on \( D^+_\gamma(A|P) \). \( \gamma(A|P; \varphi) \) predicts hypothesis of node \( A \) from the \( \alpha \) process of its parent node \( P \). In the literature, context-based methods [Torralba, 2003; Hoiem et al., 2008] can be looked as \( \gamma \) processes. Given \( D^+_\gamma(A|P) \), the MLE learning of \( \gamma \) process is defined by,

\[
\gamma(A|P; \varphi^*) = \arg \max_{\varphi} p(D^+_\gamma(A|P)|P; \varphi) \\
= \arg \max_{\varphi} \sum_{i=1}^{m} \log p(I^{(P)}_i, X(A|I^{(P)}_i)|P; \varphi) \tag{3.25}
\]

Also, \( \gamma(A|P; \varphi) \) consists of two components, one is the \( \alpha \) process of the parent node \( P \) with the parameters \( \theta_p \) and the other is the predicting model from parent node \( P \) to node \( A \) itself with parameters \( \nabla \). So, \( \varphi = (\theta_p, \nabla) \) and then,

\[
p(I^{(P)}, X(A|I^{(P)}))|P; \varphi) = p(I^{(P)}|P; \theta_p) \times p(X(A|I^{(P)}); \nabla) \tag{3.26}
\]
where $I^{(P)}$ is used to represent $I^{(P)}_i$ in general.

In the $\gamma$ process, we want to predict the location $L_A$, scale $S_A$ and orientation $O_A$ of node $A$ from the parent node $P$. So, $\nabla = (\nabla_L, \nabla_S; \nabla_O)$ and then,

$$X(A|I^{(P)}) = X(A)|X(P) = (L_A, S_A, O_A|I^{(P)})$$

The $\alpha$ process of the parent node $P$ is defined by,

$$\log p(I^{(P)}|P; \theta_P) = \alpha(P; \theta_P) \quad (3.27)$$

To predict a hypothesis of node $A$, the model used is defined by,

$$p(X(A|I^{(P)}); \nabla) = p(X(A)|X(P); \nabla) = p(L_A, S_A, O_A|I^{(P)}; \nabla) = p(L_A|I^{(P)}; \nabla_L) \times p(S_A|I^{(P)}; \nabla_S) \times p(O_A|I^{(P)}; \nabla_O) \quad (3.28)$$

Then, solving $\gamma(A|P, \varphi^*)$ depends on how the three probability distributions are modeled, $p(L_A|I^{(P)}; \nabla_L)$, $p(S_A|I^{(P)}; \nabla_S)$ and $p(O_A|I^{(P)}; \nabla_O)$. The three terms are also treated as Gaussian distributions in this chapter. So, $\nabla_L = (\mu_{L_A}, \Sigma_{L_A})$, $\nabla_S = (\mu_{S_A}, \Sigma_{S_A})$ and $\nabla_O = (\mu_{O_A}, \Sigma_{O_A})$. For example,

$$p(L_A|I^{(P)}; \nabla_L) = p(L_A|I^{(P)}; \mu_{L_A}, \Sigma_{L_A})$$

$$= \mathcal{N}(L_A; \mu_{L_A}, \Sigma_{L_A}) \quad (3.29)$$

where $\mu_{L_A}$ is the mean and $\Sigma_{L_A}$ is the covariance, estimated by the statistics in $D_+^\gamma(A|P)$.

Now, the prediction score for node $A$ from its parent node $P$ is computed by,

$$w_p^{\text{predict}} = \log p(X(A)|X(P); \nabla) \quad (3.30)$$
Then, the weight of a $\gamma$ hypothesis of node $A$ is defined by,

$$w_A^{\gamma(P)} = w_P^{\text{predict}} + w_P^{\alpha}$$  \hspace{1cm} (3.31)

Similarly, the threshold $\text{Th}(w_A^{\gamma(P)})$ of the $\gamma$ process $\gamma(A|P; \varphi)$ can be estimated in a validation $\gamma$ dataset.

The bottom panel of Fig.3.2 illustrates the learned Gaussian distributions in the $\beta$ binding and $\gamma$ prediction process for human face.

### 3.4 Evaluation Method

The proposed method is based on the discriminative power to evaluate the information contributions of the $\alpha$, $\beta$ and $\gamma$ processes individually. The method is similar to the decision tree framework [Breiman et al., 1984]. For comparisons, human performance is also studied for the three processes individually.

For simplicity of notation, denote the $\alpha$, $\beta$ and $\gamma$ processes by a testing function $\text{Tst}()$. As illustrated in Fig.3.4, the information contribution of $\text{Tst}()$, denoted by $\text{IC( Tst)}$, is measured by the uncertainty or impurity reduction after applying it on a testing dataset $D$.

The testing dataset $D = D^+ \cup D^-$ includes a set of positive samples $D^+$ and a set of negative samples $D^-$. After applying $\text{Tst}()$, two datasets are created, one is $D^+_{Tst}$ in which samples pass the testing function $\text{Tst}()$ and the other is $D^-_{Tst}$ in which samples fail. So, $D^+_{Tst}$ consist of true positives (TPs) and false positives (FPs), and $D^-_{Tst}$ include true negatives (TNs) and false negatives (FNs). Then, the information contribution of $\text{Tst}()$ is defined by,

$$\text{IC( Tst)} = 1 - \frac{\mathcal{H}(D^+_{Tst}) + \mathcal{H}(D^-_{Tst})}{\mathcal{H}(D)}$$  \hspace{1cm} (3.32)
Figure 3.4: Illustration of evaluating the information contributions of the $\alpha$, $\beta$ and $\gamma$ processes individually based on their discriminative power. The human performance is also presented for the three processes individually for comparisons. $D = D^+ \cup D^-$ are the input testing dataset including a positive sample set $D^+$ and a negative sample set $D^-$. After testing, two subsets are obtained, one is the set $D^+_{Tst}$ in which samples pass the test and the other $D^-_{Tst}$ in which samples do not pass the test. $D^+_{Tst}$ consists of TPs and FPs, and $D^-_{Tst}$ includes TNs and FNs. See texts for the calculation of information contribution.

where $\mathcal{H}(\cdot)$ represents the impurity of a dataset defined as the product of the size of the dataset (denoted by $|\cdot|$) and its entropy (denoted by $\mathcal{E}(\cdot)$),

$$\mathcal{H}(D) = |D| \times \mathcal{E}(D)$$

(3.33)

and the entropy is

$$\mathcal{E}(D) = -\frac{|D^+|}{|D|} \log \frac{|D^+|}{|D|} - \frac{|D^-|}{|D|} \log \frac{|D^-|}{|D|}$$

(3.34)

In the same way, $\mathcal{E}(D^+_{Tst})$, $\mathcal{H}(D^+_{Tst})$, $\mathcal{E}(D^-_{Tst})$ and $\mathcal{H}(D^-_{Tst})$ are calculated, and then the information contribution IC(Tst) is evaluated.

In the literature, an alternative approach for measuring Tst() is studied in [Blanchard and Geman, 2005] from some theoretical viewpoints.
3.5 Human Study

The information contribution defined in Eqn.3.32 is empirical, so the human performance of the information contributions of the \( \alpha \), \( \beta \) and \( \gamma \) processes is also evaluated individually for comparisons.

3.5.1 The Experimental Environment for Human Study

Based on the psychological toolbox [Brainard, 1997], a GUI is designed for the human study. In experiments, there are 7 human subjects with normal sights. LCD monitors are used whose brightness and contrast are adjusted for each subject adaptively. The distance between human subjects and monitors are adjusted around 50cm according to each subject’s sight. The outside light environment is also adjusted to a suitable level. In testing, clicking the enter key means the displayed sample is positive and clicking the space key means it is negative.

Observing time setting. In order to study the information contributions individually in the human study, in addition to the scale of image patch, the observing time is controlled. For the \( \alpha \) process, the observing time is less than 200ms. For the \( \beta \) and \( \gamma \) processes, the observing time is left to be controlled by the subjects themselves. At same time, the response time of each subject is recorded.

3.5.2 The Testing Data Used in Human Study

In order to reduce the amount of image data in human study, only the FPs from computer experiments are used as the negative samples for human subjects. Fig.3.6 and Fig.3.8 show some examples used in evaluations of \( \alpha \) and \( \beta \) processes of human face. The assumption is that those TNs would also be correctly classified by human subjects, which is intuitively reasonable. At the same time, each
group of data is tested by all 7 subjects to eliminate possible biases made by some subjects. The human subjects can be treated as ideal observers and their overall performance improvement against the computer can be treated as a metric in future work for the computer vision community.

3.6 Experiments

3.6.1 Junctions and Rectangles in Low-to-Middle-Level Vision

Five types of hierarchical image structures in low-to-middle-level vision are used including L-junction, cross junction, parallel line, T/Y/Arrow junction and rectangle. In the experiments, T/Y/arrow junction are treated as the same type currently due to their similarity. As illustrated in the right-bottom panel of Fig.3.3, the rectangle node is an Or-node which has two types of decompositions, one is decomposed into two groups of parallel lines and the other is decomposed into four junctions such as four L-junctions.

The data. A set of 200 natural images from the LHI image database [Yao et al., 2007] is used in which the sketches are manually labelled. Based on the manually annotated sketches, positive examples are generated for the five types of hierarchical image structures and a common set of negative examples. Some positive examples are shown in Fig.3.3.

Training and testing. For the α process, the features are generated by using first and second derivative Gaussian filters, LoG (Laplacian of Gaussian) filters, DoG (difference of Gaussian) and elongated DooG (different of offset Gaussian) filters, all with 3 scales (10 × 10, 20 × 20 and 30 × 30 pixels). The α process of line segment uses the primal sketch model [Guo et al., 2007] similar to the implicit testing used in our previous compositional boosting work [Wu et al., 2007].
α processes of the five types of hierarchical image structures use the patch-based active basis model for both shape and texture. In testing, the orientation space are searched with 15 different orientations in order to handle the rotation. For the β process, the five types of hierarchical image structures are computed by binding line segments in terms of the explicit testing on their relative locations, angles and distances between their endpoints. Rectangles are computed in two alternative ways, one is by binding two groups of parallel lines in terms of their relative locations and angles, and the other is by binding a set of incomplete (two or three) or complete (four) junctions in terms of their relative locations, angles and distances between the endpoints.
Figure 3.6: The left panel shows the information contributions of the $\alpha$ processes of nodes (i.e., head-shoulder, face, left eye, right eye, nose and mouth) in human face AoG in the human study. Five scales are tested ($8 \times 8$, $10 \times 10$, $12 \times 12$, $16 \times 16$, $20 \times 20$ and $24 \times 24$ pixels). Some positive examples for each node and some negative examples are shown in the right panel. The results show that the $\alpha$(face) process is stronger than the $\alpha$ processes of other nodes in the human face AoG.

The observation: Fig.3.5 shows the information contributions of the $\alpha$ (red lines) and $\beta$ (blue lines) processes of junctions and rectangles from the human study experiments at three scales ($10 \times 10$, $20 \times 20$ and $30 \times 30$ pixels). The results of computer experiments are shown by those small rectangles (the red ones for the $\alpha$ and the blue ones for the $\beta$ process and for clarity only the results tested with the scale $30 \times 30$ pixels are shown). The results show that the $\beta$ binding
Figure 3.7: The information contributions of the $\beta$ processes of the human face. The left panel shows the information contributions of the $\beta$ processes with 2 facial components. The right panel is for the $\beta$ processes with 3 facial components. Five scales are tested, $38 \times 38$, $50 \times 50$, $60 \times 60$, $80 \times 80$ and $100 \times 100$ pixels. Some examples are shown in Fig.3.8. In the $\beta$ process, the results show that the left eye and right eye are more informative than other facial components.

3.6.2 Human faces in High-Level Vision

The AoG of the human face consists of six nodes, head-shoulder, face, left eye, right eye, nose and mouth, as shown in Fig.3.2. In the experiments, the left and right eye node are treated as the same type due to the similarity.

The data. A set of 1000 images from the LHI database is used in which all
Figure 3.8: Some examples used in the human study of evaluating the information contribution of the $\beta$ processes of human face with 2 and 3 facial components respectively. The left panel shows some positive examples and the right panel shows some negative examples. The examples are at 100 $\times$ 100 pixels for illustration.

the six nodes are at good resolution and the parse graphs are manually labelled (see an example in Fig.3.1). The training data are generated based on the parse graph.
Training and testing. For the $\alpha$ process, the Gabor wavelets are used. The learned $\alpha$, $\beta$ and $\gamma$ processes are shown in Fig.3.2. Here, five scales are tested for the $\alpha$ process ($8 \times 8, 10 \times 10, 12 \times 12, 16 \times 16, 20 \times 20$ and $24 \times 24$ pixels), five scales for $\beta$ process ($38 \times 38, 50 \times 50, 60 \times 60, 80 \times 80$ and $100 \times 100$) and one scale for the $\gamma$ process of human face ($32 \times 32$).

The observation: From the results, it is observed that the $\alpha$ process of the human face node is stronger than those of the other node in the human face AoG. The information contributions are shown in Fig.3.6.
CHAPTER 4

Integrating BU/TD Processes by Breadth-First-Search (BFS)

This chapter presents a method of integrating the BU/TD (i.e., α-β-γ) processes by Breadth-First-Search (BFS) to improve accuracy performance in object parsing using AoG formulated under Bayesian framework. Based on the information contributions of the three processes evaluated in Chapter 3, all the α(A), β(A) and γ(A) processes contribute to compute node A from images in complementary ways which depend on the scale and occlusion conditions (see Fig.1.1 and Fig.2.4). Given an input image, the objective of object parsing with AoG is to construct the parse graph for each appeared object instance of node A (e.g., the human face) on the fly where the specific situations of node A (i.e., locations, scales and orientations) are unknown in advance and to be inferred. In this chapter, object parsing with AoG is formulated under the Bayesian framework, and then the Bayesian posterior of a parse graph is factorized to connect the α-β-γ processes to Bayesian inference and dynamic programming implementation. For robust inference, the α(A), β(A) and γ(A) processes are explicitly integrated and a greedy pursuit algorithm is proposed for object parsing by BFS according to the AoG. With experiments on human face parsing and hierarchical image structure parsing, the results show performance improvement from the integration in terms of the ROC comparisons.
4.1 Bayesian Formulation

An AoG represents the object parsing grammar. Given an input image $I_\Lambda$, it contains an unknown number $K$ object instances with unknown locations, scales, orientations and occlusions. The goal of object parsing using AoG is to construct a parse graph for each object instance in $I_\Lambda$ on the fly, denoted by $pg_k$ ($k = 1, \cdots, K$). For example, for the human face parsing, Fig.1.1 shows a typical testing image and the left-bottom panel in Fig.4.1 shows a number of inferred parse trees of human face instances.

Then, the goal of object parsing using AoG is to seek a world representation $W$ for image $I_\Lambda$, defined by,

$$W = (K, \{pg_k\}_{k=1}^K) \quad (4.1)$$

Under the Bayesian framework, the optimal $W^*$ is inferred by maximizing a posterior probability $p(W|I_\Lambda)$,

$$W^* = \arg \max_{W \in \Omega} p(W|I_\Lambda) = \arg \max_{W \in \Omega} p(W) \times p(I_\Lambda|W) \quad (4.2)$$

where $\Omega$ is the solution space.

4.1.1 The Prior Probability

In this chapter, the prior probability $p(W)$ is defined by,

$$p(W) = p(K) \times p(\{pg_k\}_{k=1}^K) = p(K) \times \prod_{k=1}^K p(pg_k) \quad (4.3)$$

where $p(K)$ is the prior distribution for the number of object instances (e.g., an exponential model $p(K) \propto \exp\{-\lambda_0 K\}$ is used in this chapter) and $p(pg_k)$ is the prior model of a parse graph defined by Eqn.2.3. In object parsing, constructed
parse graphs of object instances compete each other to explain away the image data so \( p(\{pg_k\}_{k=1}^K) \) is factorized into \( \prod_{k=1}^K p(pg_k) \).

### 4.1.2 The Likelihood Model

Let \( \Lambda_{pg_k} \) be the image lattice occupied by the parse graph \( pg_k \) (1 \( \leq k \leq K \)). Denote \( \Lambda_{fg} = \bigcup_{k=1}^K \Lambda_{pg_k} \) as the foreground lattice and \( \Lambda_{bg} = \Lambda \setminus \Lambda_{fg} \) as the remaining background lattice. Then, \( I_\Lambda = (I_{\Lambda_{fg}}, I_{\Lambda_{bg}}) \). Let \( q(I) \) be the generic background model which will be made implicit in the following derivation. The image likelihood model \( p(I_\Lambda|W) \) is defined by,

\[
p(I_\Lambda|W) = p(I_{\Lambda_{fg}}, I_{\Lambda_{bg}}|W) = p(I_{\Lambda_{fg}}|W) \times q(I_{\Lambda_{bg}})
= p(I_{\Lambda_{fg}}|W) \times q(I_{\Lambda_{bg}}) \times \frac{q(I_{\Lambda_{fg}})}{q(I_{\Lambda_{fg}})}
= q(I_\Lambda) \times \frac{p(I_{\Lambda_{fg}}|W)}{q(I_{\Lambda_{fg}})}
= q(I_\Lambda) \times \prod_{k=1}^K \frac{p(I_{\Lambda_{pg_k}}|pg_k)}{q(I_{\Lambda_{pg_k}})}
\]

(4.4)

where \( p(I_{\Lambda_{pg_k}}|pg_k) \) represents the likelihood of the image domain \( \Lambda_{pg_k} \) is explained by the parse graph \( pg_k \) and \( q(I_{\Lambda_{pg_k}}) \) is the likelihood of explaining the domain \( \Lambda_{pg_k} \) as background. These models compete with each other in object parsing and \( \frac{p(I_{\Lambda_{fg}}|W)}{q(I_{\Lambda_{fg}})} \) is also factorized due to the competition among parse graphs.

So, Eqn.4.2 can be reproduced as,

\[
W^* = \arg \max_{W \in \Omega} p(K) \times q(I_\Lambda) \times \prod_{k=1}^K \left[ p(pg_k) \times \frac{p(I_{\Lambda_{pg_k}}|pg_k)}{q(I_{\Lambda_{pg_k}})} \right]
= \arg \max_{W \in \Omega} p(K) \times \prod_{k=1}^K \left[ p(pg_k) \times \frac{p(I_{\Lambda_{pg_k}}|pg_k)}{q(I_{\Lambda_{pg_k}})} \right]
\]

(4.5)

where \( q(I_\Lambda) \) is an unknown constant and then can be omitted in the maximization.
4.2 Object Parsing by Bayesian Inference

In the literature, there are several ways to infer $W^*$ in Eqn.4.5, such as the data-driven Markov chain Monte Carlo (DDMCMC) method used in [Tu and Zhu, 2002]. In this chapter, the goal is to pursue object instances appearing in an input image and construct corresponding parse graphs. Often, the number of object instances $K$ is typically not too large (e.g., less than 10). So, BFS is adopted to pursue parse graphs sequentially by maximizing Eqn.4.5 in a greedy manner. The pursuit inference algorithm integrates the $\alpha \beta$ and $\gamma$ processes and includes two aspects: (i) Generating proposals (hypotheses) for possible parse graphs and (ii) Verifying parse graph proposals in a greedy pursuit manner.

4.2.1 Factorizing the Bayesian Posterior

Parse graphs are constructed sequentially based on Eqn.4.5 starting from an initial empty solution $W_0 = \emptyset$,

$$W_0 = \emptyset \rightarrow W_1 \rightarrow \cdots \rightarrow W_k \rightarrow \cdots \rightarrow W_K = W^*$$  \hspace{1cm} (4.6)

At each step a new parse graph is pursued and then at the step $k \geq 1$, denote by $\Lambda_{fg,k} = \bigcup_{i=1}^{k-1} \Lambda_{pgi}$ the current foreground domain and by $\Lambda_k = \Lambda \setminus \Lambda_{fg,k}$ the remaining image lattice, then the $k$-th parse graph $pg_k$ is computed by,

$$pg^* = \arg \max_{pg \in \Omega_{pg}} p(pg) \times p(I_{\Lambda_k} | pg)$$  \hspace{1cm} (4.7)

where the subscript $k$ in $pg_k$ is omitted in the derivation for simplicity when there is no confusion. $\Omega_{pg}$ is the proposal space of parse graphs.
Similar to derive Eqn.4.4, \( p(I_{\Lambda_k} | pg) \) is computed by,

\[
p(I_{\Lambda_k} | pg) = p(I_{\Lambda_{pg}}, I_{\Lambda_{k+1}} | pg) = p(I_{\Lambda_{pg}} | pg) \times q(I_{\Lambda_{k+1}}) = q(I_{\Lambda_k}) \frac{p(I_{\Lambda_{pg}} | pg)}{q(I_{\Lambda_{pg}})} \quad (4.8)
\]

So, Eqn.4.7 can be rewritten as,

\[
p^* = \arg \max_{pg \in \Omega_{pg}} p(pg) \frac{p(I_{\Lambda_{pg}} | pg)}{q(I_{\Lambda_{pg}})} = \arg \max_{pg \in \Omega_{pg}} [\log p(pg) + \log \frac{p(I_{\Lambda_{pg}} | pg)}{q(I_{\Lambda_{pg}})}] \quad (4.9)
\]

which is consistent with Eqn.4.5.

Recall that the prior probability \( p(pg) \) is defined in Eqn.2.3. For object categories with roughly rigid configuration such as the human face, there are no occlusion among different nodes at the same layer in a parse graph so that the likelihood ratio \( \frac{p(I_{\Lambda_{pg}} | pg)}{q(I_{\Lambda_{pg}})} \) can be factorized with respect to Eqn.2.6,

\[
\log \frac{p(I_{\Lambda_{pg}} | pg)}{q(I_{\Lambda_{pg}})} = \sum_{t \in V_{\Lambda_{pg}}} \log \frac{p(I_{\Lambda_t} | t)}{q(I_{\Lambda_t})} \quad (4.10)
\]

Without loss of generality, consider the AoG illustrated in Fig.2.1 where node \( A \) represents the object of interest (e.g., the human face) and \( V_{\text{and}} = \{ P, A, C_1, C_2, C_3 \} \). Furthermore, consider a parse graph \( pg \) with \( V_{\text{and}}^{pg} = \{ P, A, C_1, C_2 \} \) and \( V_T^{pg} = \{ t_P, t_A, t_{C_1}, t_{C_2} \} \). By using Eqn.2.3, the prior probability \( p(pg) \) is defined by,

\[
\log p(pg) = \log p(A | O) + \log p(X(A) | X(P)) + \sum_{i=1}^{2} \log p(X(C_i) | X(A)) + \log p(X(C_1), X(C_2)) - \log Z \quad (4.11)
\]
Then, by combining Eqn.4.10 and Eqn.4.11, Eqn.4.9 can be rewritten as,

\[ pg^* = \arg \max_{pg} \left\{ \log p(A|O) + \right. \]
\[ \log \frac{p(I_{N_A}|t_A)}{q(I_{M_A})} + \left. \right. \]
\[ \left. \left[ \sum_{i=1}^{2} \log \frac{p(I_{N_{C_i}}|t_{C_i})}{q(I_{M_{C_i}})} \right] + \sum_{i=1}^{2} \log p(X(C_i)|X(A)) \right. \]
\[ \left. + \log p(X(C_1), X(C_2)) \right] \]

where \( p(A|O) \) represents the inference of alternative structures (e.g., a mixture component). \( p(X(C_i)|X(A)) \) in the binding model represents local deformation penalty of part \( C_i \) with respect to node \( A \), and \( p(X(C_1), X(C_2)) \) represents mutual compatibility. Often, for simplicity in inference, parts only connect to the parent node, i.e., not considering \( p(X(C_1), X(C_2)) \).

From Eqn.4.12, the \( \alpha \), \( \beta \) and \( \gamma \) processes are integrated explicitly in pursuing parse graphs. Denote by \( w_A^\alpha \), \( w_A^{\beta(c)} \) and \( w_A^{\gamma(P)} \) as the weights computed from the \( \alpha \), \( \beta \) and \( \gamma \) processes respectively,

\[ w_A^\alpha = \log \frac{p(I_{N_A}|t_A)}{q(I_{N_A})} \]
\[ w_A^{\beta(c)} = \sum_{i=1}^{2} \left[ \log \frac{p(I_{N_{C_i}}|t_{C_i})}{q(I_{M_{C_i}})} + p(X(C_i)|X(A)) \right] + \log p(X(C_1), X(C_2)) \]
\[ w_A^{\gamma(P)} = \log \frac{p(I_{N_P}|t_P)}{q(I_{N_P})} + \log p(X(A)|X(P)) \]

which are specified by the MLE learning in Sec.3.3 in Chapter 3. Then, Eqn.4.12
is reproduced as,

\[
pg^* = \arg \max_{pg \in \Omega_{pg}} \{ \log p(A|O) + \underbrace{w^{\alpha}_{A}}_{\alpha(A) \text{ process}} + \underbrace{w^{\beta(c)}_{A}}_{\beta(A) \text{ process}} + \underbrace{w^{\gamma(P)}_{A}}_{\gamma(A) \text{ process}} \} \quad (4.16)
\]

4.2.2 Connecting the α-β-γ Processes to Bayesian Inference

This section introduces the formal specifications of the α, β and γ processes by connecting the general identifications and definitions of the three processes in Chapter 2 with the Bayesian inference in terms of Eqn.4.12.

**The α process in Bayesian inference.** The α process detects node A by applying a log-likelihood ratio test, \( \log \frac{p(I_{A_{|tA}})}{q(I_{A_{|tA}})} \) (Eqn.4.13), directly based on image features when node A is at middle resolution without occlusion. The α process can be viewed as either bottom-up (feature classifiers such as the Adaboost method) or top-down (template matching such as the active basis model) inference process. For each And-node \( A \in V_{and} \), the α process, \( \alpha(A; \theta) \), is instantiated by a corresponding terminal node \( t_A \in V_T \), where \( \theta \) is a set of parameters. For example, in Fig.2.1, \( \alpha(A; \theta) \) is specified in term of,

\[
\alpha(A; \theta) : t_A \rightarrow A \text{ and } x(t_A) \Rightarrow X(A) \quad (4.17)
\]

where \( t_A \rightarrow A \) is calculated by \( w^\alpha_A \) in Eqn.4.13 based on Eqn.3.11, and \( x(t_A) \Rightarrow X(A) \) is used to activate the γ processes of node A’s child nodes, \( p(X(C_i)|X(A)) \), in Eqn.4.12 and the β process of node A’s parent node, \( p(X(P)|X(A)) \).

**The β process in Bayesian inference.** The β process computes node A by applying a bottom-up binding test, for example \( \log p(X(C_1), X(C_2)) \) (in Eqn.4.14), of its child nodes \( c = (C_1, C_2) \) which have been detected in a given step based
Figure 4.1: A running example of pursuing human faces and constructing corresponding parse graphs in a typical image by integrating the $\alpha$, $\beta$ and $\gamma$ processes. (a) shows the proposals from all the $\alpha$ processes. (b) illustrates how to generate parse graph proposals. (c) shows the results of pursuing object instances and constructing their parse graphs on the fly.
Figure 4.2: (a) shows all candidates from $\alpha$ processes, i.e., "things and stuffs". (b) shows the candidates after thresholding.
Figure 4.2: (c) shows the constructed parse graph of the first pursued face. (d) shows the result for the second pursued face.
Figure 4.2: (e) shows all constructed parse graphs. (f) shows remaining stuffs after pursuing.
on the log-likelihood ratio tests of their own $\alpha$ processes, $w_{C_i}^{\alpha} = \log \frac{p(I_{t_{C_i}} | t_{C_i})}{q(I_{t_{C_i}})}$.

The $\beta$ process handles the situation in which node $A$ is at high resolution but with occlusion (the occlusion disable the $\alpha(A)$ process). Let $V_{\text{and}}^\text{ch} \subset V_{\text{and}}$ be the set of And-nodes which have children. For each node $A \in V_{\text{and}}^\text{ch}$, the $\beta$ process of node $A$ can be defined, denoted by $\beta(A|c; \phi)$ where $c \subseteq ch(A)$ and $\phi$ is a set of parameters. Given different $c$’s, we obtain different $\beta$ processes for node $A$.

Consider $c = (C_1, C_2)$ in Fig.2.1, $\beta(A|c; \phi)$ is specified by,

$$\beta(A|c; \phi) : t_{C_1} \rightarrow C_1 \text{ and } x(t_{C_1}) \Rightarrow X(C_1)$$

$$t_{C_2} \rightarrow C_2 \text{ and } x(t_{C_2}) \Rightarrow X(C_2)$$

$$(C_1, C_2) \rightarrow A \text{ and } (X(C_1), X(C_2)) \Rightarrow X(A)$$

where $t_{C_i} \rightarrow C_i$ are calculated by $w_{C_i}^{\alpha} (i = 1, 2)$, and $(X(C_1), X(C_2)) \Rightarrow X(A)$ will activate the $\beta$ binding process of node $A$. $w_{A}^{\beta(c)}$ is computed based on Eqn.3.24. The $\beta(A|c; \phi)$ will, in turn, activate the $\gamma$ processes of the other child nodes of node $A$ and the $\beta$ process of node $A$’s parent node. This procedure is activated recursively in testing.

The $\gamma$ process in Bayesian inference. The $\gamma$ process computes node $A$ from images by applying a top-down prediction test, $\log p(X(A)|X(P))$ (in Eqn.4.15), from its parent node $P$ which has been already detected in a given step based on the log-likelihood ratio test of the $\alpha$ process of node $P$, $w_{P}^{\alpha} = \log \frac{p(I_{t_{P}} | t_{P})}{q(I_{t_{P}})}$. The $\gamma$ process handles the situation in which node $A$ is under very low resolution so both $\alpha(A)$ and $\beta(A)$ are disabled. Let $V_{\text{and}}^\text{prt} \subset V_{\text{and}}$ be the set of And-nodes which have parent node(s). For each node $A \in V_{\text{and}}^\text{prt}$, we can define its $\gamma$ process, denoted by $\gamma(A|P; \varphi)$ where $P \in \text{prt}(A)$ is a parent node and $\varphi$ is a set of parameters.
Similarly, in Fig. 2.1, \(\gamma(A|P; \varphi)\) is specified by,

\[
\gamma(A|P; \varphi) : t_P \rightarrow P \text{ and } x(t_P) \Rightarrow X(P)
\]

\[
P \rightarrow A \text{ and } X(P) \Rightarrow X(A)
\]

where \(t_P \rightarrow P\) is calculated by \(w^\alpha_P\) and \(X(P) \Rightarrow X(A)\) will activate the \(\gamma\) process of node \(A\). \(w^\gamma_P\) is computed based Eqn. 3.31, and activate the inference process recursively.

### 4.3 Integrating the \(\alpha\)-\(\beta\)-\(\gamma\) Processes by BFS

The BFS pursuit algorithm is straightforward based on Eqn. 4.12. Fig. 4.1 and Fig. 4.2 show a running example of human face parsing by the proposed algorithm. On the whole, the algorithm first runs all \(\alpha\) processes (see the top panel in Fig. 4.1) and applies thresholds to obtain candidates for each node (see Fig. 4.2 (a) and (b)). Then, to pursue object instances of node \(A\), the algorithm recursively runs all \(\beta\) processes and \(\gamma\) processes to do bottom-up binding and top-down prediction and then generate the parse graph proposals ((see Fig. 4.2 (c) to (f))). The pursuit is based on a proposed pursuit index defined by integrating the scores from the \(\alpha\)-\(\beta\)-\(\gamma\) processes. The parse graph for each pursued instance of node \(A\) is constructed by retrieving all the used \(\alpha\) candidates (see the bottom panel in Fig. 4.1). the algorithm is summarized in Alg. 1.

**Generating Proposal Space \(\Omega_{pg}\)**. In object parsing, in order to find the object instances which appear at different locations, scales and orientations, the three corresponding spaces need to be searched to find possible proposals. For searching the scale space, the image pyramid for \(I_A\) is created using a certain down-sampling factor (e.g., 0.9 used in experiments in this chapter)
until the image size is smaller than the minimum of sizes required by $\alpha$ processes (i.e., the minimum of sizes of learned active basis templates). Denote by \( \{ I_{\Lambda_0} = I_\Lambda, I_{\Lambda_1}, \cdots, I_{\Lambda_L} \} \) a $L$-layer image pyramid. For exploring different orientations, either different image pyramids are created for different orientations if the learnt $\alpha$ processes are not rotatable (due to the features used in the process), or different $\alpha$ processes are created for different orientations by transforming the learnt one (such as the active basis models used in this chapter). Then, the sliding window method is adopted to search all locations in the image pyramids for all active basis templates, i.e., all the $\alpha$ processes are tested first. By applying the thresholds, a set of candidates are obtained for each node in the AoG (which could be empty for some nodes due to for example occlusions). The parse graph proposal space $\Omega_{pg}$ are implicitly generated based on all the candidates. Recall that fixed relative scales are used in training the $\alpha(A)$, $\beta(A)$ and $\gamma(A)$ processes. So, for a hypothesis of node $A$ (such as the human face), the locations, scales and orientations of other nodes (such as head-shoulder and eyes) are roughly specified in terms of the learned $\beta$ binding models $p(X(C_i)|X(A))$ and $\gamma$ prediction models $p(X(A)|X(P))$. Then, an open list is created containing complete or partial parse graphs which may overlap and therefore compete each other to explain the corresponding image domain. The open list are explored based on the pursuit index (defined below) to do the proposal verification.

**Pursuit index.** For each proposal, its total weight is computed by integrating the scores from the $\alpha$-$\beta$-$\gamma$ processes,

\[
    w_A = w_A^\alpha + w_A^\beta(c) + w_A^\gamma(P)
\]

which is the pursuit index used for proposal verification. In practice, a non-max suppression step will be used to declare detections.
Algorithm 1: Integrating $\alpha$-$\beta$-$\gamma$ Processes for Object Parsing with AoG

**Input**: An image $I_A$ and an AoG $G$

**Output**: Parse graphs $pg_i$ ($i = 1, \cdots, K$)

1. **$\alpha$ map generation**: $I_{\alpha(U)}$, $\forall U \in V$ and run $\alpha(U; \theta_U)$ and compute the weight $w_{\alpha U}$.

2. **$\alpha$ hypotheses generation**: decreasingly-ordered open list $OP(U)$ from $I_{\alpha(U)}$ apply thresholds $Th(w_{\alpha U})$ and local inhibitions.

3. **$\beta$ bindings and merging**.
   (i) run $\beta(A \mid c; \phi)$ and compute the weight $w_{\beta A}^{c(e)}$
   (ii) apply $Th(w_{\beta A}^{c(e)})$ to generate $\beta(A)$ hypotheses and insert them into $OP(A)$;
   (iii) merge with compatible $\alpha(A)$ hypotheses and compute weight $w_{\alpha A}^\alpha + w_{\beta A}^{\beta(e)}$.

4. **$\gamma$ predictions and merging**.
   (i) run $\gamma(A \mid P; \varphi)$ and compute the weight $w_{\gamma A}^{(P)}$
   (ii) apply $Th(w_{\gamma A}^{(P)})$ to generate $\gamma(A)$ hypotheses and insert them into $OP(A)$;
   (iii) merge with compatible $\alpha(A)$, $(\alpha + \beta)(A)$ hypotheses and compute weight $w_{\alpha A}^\alpha + w_{\gamma A}^{(P)}$ or $w_{\alpha A}^\alpha + w_{\beta A}^{\beta(e)} + w_{\gamma A}^{(P)}$.

5. **Object pursuing and parsing**. In $OP(A)$, pursue node $A$ according to $w_A$, construct $pg_i$’s by retrieving all the $\alpha$ hypotheses. Stop pursuing based on $Th(w_A)$.
4.3.1 Connection with Dynamic Programming (DP)

The proposed BFS pursuit algorithm can be treated as Max-Product DP like algorithm with partial DP tables due to the thresholding step used for each process. In DP, all the $\alpha$ processes are also computed first, and then for each $\beta$ binding relations and $\gamma$ prediction model, a DP table is created to memoize the transformed contribution from a child node or the parent node. The final score for a parse graph is the same form as Eqn.4.20 but without early decisions (i.e., thresholding) made for each process, and the parse graphs are retrieved with all nodes being placed even if some are occluded or missing. The proposed pursuit algorithm attempts to purse object instances which may have complete or partial parse graphs, and to compare the performance of different types of combinations of the three processes explicitly.

4.4 Experiments

4.4.1 Performance Comparisons

According to Eqn.4.20, the performance of computing node $A$ by different integrations of $\alpha$-$\beta$-$\gamma$ processes are compared explicitly. For example, ROCs are plotted based on $w_A^{\alpha}$, $w_A^{\alpha} + w_A^{\beta(e)}$, $w_A^{\alpha} + w_A^{\gamma(P)}$ and $w_A$ respectively. Some comparison results are shown in Fig.4.4 for junction and rectangle parsing and Fig.4.5 for human face parsing. These ROCs show that how and how much the integration improves performance for different object categories. The ROC comparisons are consistent with the evaluated information contributions in experiments in Chapter 3.
4.4.2 Rectangle Parsing

A set of 50 images are tested which includes 30 city scene images and 20 office scene images. A running example is shown in Fig.4.3 and more examples are shown in Fig.4.4. From the ROC comparisons in Fig.4.4, the results show that the \( \beta \) processes of junctions and rectangles in low-to-middle-level vision dominate with much performance improved against the \( \alpha \) processes.

4.4.3 Human Face Parsing

A set of 500 images is tested in which more than half of human face instances are with different occlusions or at very low resolution. A running example of the human face pursuit is shown in Fig.4.1 and more examples are shown in Fig.4.5. From the ROC comparisons in Fig.4.5, the results show that for human face, its \( \alpha \) process works better than those of its child nodes such as eyes and nose and its parent node such as head-shoulder.
Figure 4.3: A running example of pursuing junctions and rectangles in a typical image by integrating the $\alpha$, $\beta$ and $\gamma$ processes. The left-top is the original input image. The middle-top is the edge probability map and the right-top shows the detected corners. The images in the second and the third row show the detected results of different kinds of junctions with the type name shown in the left-top in each image. The left-bottom shows the detected rectangles. The middle-bottom shows the final sketch by merging all detected results. Compared with the canny results shown in the right-bottom image, we can see that the final sketch obtained by the proposal algorithm is better.
Figure 4.4: The top panel show more results of rectangle pursuing. The bottom panel shows the ROC comparisons of the $\alpha$ process and the integration of the $\alpha$, $\beta$ processes for junctions and rectangle. Those black pentagrams show the performance by humans.
Figure 4.5: The top panel shows more results of human face pursuing. The bottom panel shows the ROC comparisons of the $\alpha$ process and different integrations of the $\alpha$, $\beta$ and $\gamma$ processes. Those black pentagrams show the performance by humans.
CHAPTER 5

Learning Near-Optimal Cost-Sensitive Decision Policies

To advance on-line computational efficiency of a learnt computing process given goal-guided allowable bounds on accuracy performance, this chapter presents a method of learning near-optimal decision policies for computing processes whose scoring functions are factorizable in the sequential testing manner (e.g., Adaboost [Viola and Jones, 2004], the active basis model [Wu et al., 2010] and the deformable part-based model [Felzenszwalb et al., 2010]). Concretely, a given scoring function is factorized into a sum of $N$ basic terms (such as the weak classifiers in a learnt Adaboost strong classifier) with a fixed order and then is organized into two parts during on-line computing, the to-arrive scores and the to-go scores. The decision policy consists of $N$ pairs of thresholds of the sequential to-arrive scores for acceptances and rejections respectively such that early decisions can be made to reduce runtime and the given allowable bounds on accuracy performance (i.e., FNR and FPR) are guaranteed. To learn the decision policy, a risk function is defined which takes into account the computational cost, the FN and FP loss costs or the FPR and FNR (accounting for cost-sensitive vision goals). Then, the near-optimal decision policy is learnt by minimizing the risk function using dynamic programming to exploit the histograms of the to-arrive scores in positive and negative datasets. The scoring functions of the computing processes...
of terminal nodes and And-nodes are factorizable as defined in Chapter 3 and their decision policies are learned in experiments. The learnt decision policy can be used to estimate the heuristic of the corresponding computing process and the estimated heuristic will be used in the scheduling algorithm in Chapter 6.

5.1 Background

Cascade, Coarse-to-Fine and Branch-and-Bound Strategies. In most existing works on object detection and parsing, the sliding window technique is widely used for on-line computing. Because of its computationally expensive cost, there are some improvements proposed in recent literature. During on-line computing, most of sliding windows belong to background, which can be ruled out with some simple and computationally cheap testings. Based on this observation, cascade boosting methods [Viola and Jones, 2004] train a multi-stage classifier in which computationally expensive testing are only applied on those windows survived so that online computing time is reduced. Coarse-to-fine strategies [Blanchard and Geman, 2005; Amit et al., 2004; Gangaputra and Geman, 2006; Sahbi and Geman, 2006; Sznitman and Jedynak, 2010] organize all the testings in a hierarchy such as decision tree based on a top-down coarse-to-fine partition of the pose space of the object to be detected, and from the root node to each leaf node, each testing is selected in terms of keeping the false negative (type II error) equal to zero. In essence, both approaches work in an attentional manner, shifting computational expensive testing onto those windows survived from earlier simple testing dedicated for backgrounds so more promising for targets appearing. Instead of working in a sliding window way, efficient search algorithms based on branch-and-bound directly search sub-windows of object instances through bounding the quality of a testing function (such as a trained
classifier) from the upper and the lower for any given sub-window. The key is to create the lower and upper bounds, and current approaches [Lampert et al., 2009] only adapt well for classifiers trained in terms of bag-of-word based features.

**Sequential Hypothesis Testing.** Sequential hypothesis testing is a method of reducing the number of samples in on-line computing by connecting the rejection and acceptance thresholds of the cumulative scores to the specified the FNR (i.e., type II error rate) and FPR (i.e., type I error rate). The widely used scheme is based on Wald’s Sequential Probability Ratio Test (SPRT) [Wald, 1947] which can be treated as a sequential extension of the Neyman-Pearson lemma that the likelihood ratio test is the most powerful one for a given significance level and a threshold [Neyman and Pearson, 1933]. In the literature of computer vision, SPRT is often used for point hypotheses or different variations of SPRT are adopted, such as truncated SPRT [Seigmund, 1985], etc. In [Pele and Werman, 2008], to address fast computing in the problem of pattern matching using Hamming distance, a Bayesian sequential hypothesis testing method is proposed by exploiting a prior distribution of the Hamming distance to address SPRT with composite hypotheses. The proposed method of learning cost-sensitive decision policies in this chapter is related to [Pele and Werman, 2008], but with two main difference: (i) The scoring function is more general than the Hamming distance function (which is a step-wisely increasing function of the number of samples tested); and (2) Instead of introducing a generic prior distribution which is not straight forward to compute for the scoring functions considered in this thesis, the histograms of training scores on positive and negative dataset are exploited to learn the decision policy.
5.2 Problem Formulation

5.2.1 Factorizing a Scoring Function

Denote by $f(x)$ the scoring function of a learnt computing process which is factorizable in the sequential testing manner and by $\tau$ its threshold estimated in training. Concretely, $f(x)$ is factorized into $N$ terms $f_i(x)$ ($i = 1 \cdots , N$) where the order $\{f_1(x), \cdots , f_N(x)\}$ is based on either the natural order obtained in learning (such as the order of weak classifiers in Adaboost or the pursued basis in active basis model) or a specified and fixed order after learning (e.g., the order sorted based on the cost/power ratios as done in [Blanchard and Geman, 2005]). Then, under the sequential testing method in on-line computing, $f_i(x)$'s are organized into two parts: the to-arrive score, denoted by $g(x,n)$, computed by testing $n$ out of $N$ terms in $f(x)$, and the to-go score, denoted by $h(x,\neg n)$, representing the score of the remaining terms but not tested yet (where $\neg n$ represents the set of indexes of remaining terms, i.e., $\{n+1, \cdots , N\}$). So, $f(x)$ can be written as,

$$f(x) = \sum_{i=1}^{N} f_i(x) = \sum_{i=1}^{n} f_i(x) + \sum_{j=n+1}^{N} f_j(x) \tag{5.1}$$

$$\triangleq g(x,n) + h(x,\neg n)$$

Examples. In Adaboost, a strong classifier predictor consists of linearly combined $N$ weak classifiers $f_i(x) = \lambda_i b_i(x)$ (where $\lambda_i$ is the learnt weight for the binary weak classifier $b_i(x) : x \rightarrow \{-1, 1\}$). In SVM, a classifier consists of $N$ support vectors $f_i(x) = w_i K(x_i, x)$ (where $K(\cdot , \cdot)$ represents kernel function and $x_i$ is a support vector). In the active basis model [Wu et al., 2010], the matching score is the sum of log-likelihood ratios of $N$ pursued active basis $f_i(x) = \lambda_i r_i(x) - \log z_i$ (where $r_i(x)$ represents Gabor filter response after sigmoid transformation and $z_i$ is the learnt normalization constant).
5.2.2 The Definition of a Decision Policy

To advance the runtime of $f(x)$ under sequential testing, the number $n$ of tested terms in $g(x,n)$ should be as small as possible, i.e., an early decision is made based on the to-arrive score $g(x,n)$, but still satisfying the constraint that accuracy performance should be obtained as close with given allowable bounds as possible. To that end, the decision policy for $f(x)$ is defined and the near-optimal one is learnt by exploiting the statistics of $g(x,n)$ jointly in this thesis. Denote by $\Pi_N$ the decision policy for $f(x)$ which is defined by a $N$-tuple,

$$\Pi_N = (\tau_1, \cdots, \tau_N)$$

and each element $\tau_n$ represents a pair of thresholds,

$$\tau_n = \{\tau_{n,1}, \tau_{n,2} \mid \tau_{n,1} \leq \tau_{n,2}\} \text{ for } n = 1, \cdots, N - 1$$

$$\tau_N = \{\tau_{N,1}, \tau_{N,2} \mid \tau_{N,1} = \tau_{N,2}\}$$

where $\tau_{N,1} = \tau_{N,2}$ is enforced in $\tau_N$ because $g(x,N) = f(x)$ is the full score and then the decision is assumed to be made.

Then, during on-line sequential testing for $x$, there are three actions:

- Accept $x$, if $g(x,n) \geq \tau_{n,2}$
- Reject $x$, if $g(x,n) < \tau_{n,1}$
- Continue to compute $g(x,n + 1)$, if $n < N$ and $\tau_{n,1} \leq g(x,n) < \tau_{n,2}$.

Furthermore, at each step $n$, $\tau_{n,1}$ can take the special value $\tau_{n,1} = -\infty$ meaning that no rejections should be decided, and similarly $\tau_{n,2} = +\infty$ indicating that no acceptances could be made.

To learn the decision policy, all the to-arrive scores $g(x,n)$ for $n = 1 \cdots, N$ on positive and negative samples are exploited jointly. Let $D = D^+ \cup D^-$ be
the training datasets for $f(x)$ consisting of the positive and negative training dataset (e.g., the $\alpha$ process positive and negative training data $D^+_\alpha$ (Eqn. 3.2) and $D^-_\alpha$ respectively). Denote the set of to-arrive scores at step $n$ computed on the training set by

$$G(n) = \{g(x,n); \forall x \in D\} \text{ for } n \in [1,N]$$

which is obtained as the by-products during learning $f(x)$. Then, $G(n)$ is discretized into $K_n$ bins (where $K_n$’s can take either the same value or different values) and let $K = \max(K_1, \cdots, K_n)$. Denote by $k$ be the index of a $g(x,n)$ (where $x \in D, n \in [1,N]$, and the subscript $n$ is used in $k_n$ when necessary) and then $k \in [1,K_n]$, and for notation simplicity still denote by $(\tau_{n,1}, \tau_{n,2})$ the indexes of the thresholds. Then, the decision policy $\Pi_N$ can be treated as a $N \times K$ matrix. For example, Fig. 5.1 shows the learnt decision policy for the human face Adaboost classifier consisting of 250 boosted weak classifiers which are factorized into 10 subsets (i.e., $f_i(x)$’s) in terms of the boosted order, where the blue line represents the acceptance line and the yellow one the rejection line, and the red and blue curves are the histograms for positive and negative to-arrive scores $G(n)$ respectively (dashed one for the whole $G(n)$ and solid one for the subset continued).

**Statistical Fluctuation.** In practice, $G(n)$ is augmented by incorporating bootstrapped to-arrive scores which are sampled from the Gaussian models estimated using the original positive and negative to-arrive scores respectively to account for the statistical fluctuations. For notational simplicity, $G(n)$ is still used to denote the augmented set.
Figure 5.1: Illustration of the decision policy learnt for a human face Adaboost classifier which consists of 250 boosted weak classifiers divided into 10 subsets (i.e. \( f_i(), \ i = 1, \cdots, 10 \)). The green and yellow lines represent the acceptance and rejection lines respectively. Each row shows the positive (red) and negative (blue) histograms of to-arrive scores in \( G(n) \) where the dashed curves are plotted for the whole \( G(x) \) and the solid ones for the subset continued. For clarity, all rows use the same number of bins (i.e., \( K_n \)'s are the same).
Figure 5.2: The vertical solid and dashed red lines show the paths of positive samples randomly selected from the “best” and “worst” subsets in terms of the final scores $f(x)$’s. The vertical solid and dashed blue lines are paths of negative samples. From the paths, together with the statistics of the band width of paths (i.e., the difference between successive bin indexes) shown in Fig. 5.3, it motivates the early decisions in the decision policy.
Figure 5.3: The plots show the statistics, of the maximal and mean band width and corresponding standard derivations, of paths of positive and negative examples traversing in the decision policy matrix.

5.2.3 The Data Flow Distributed by a Decision Policy

After discretized, the set of all to-arrive scores, denoted by $G = \{G(n); n \in [1, N]\}$, is distributed into the $N \times K$ entries in $\Pi_N$, and each example $x \in D$ corresponds to $N$ terms in $G$ (i.e., $\{g(x, n); n = 1, \cdots, N\}$) each of which falls into one bin at one row. Fig. From the perspective of data flow distributed in $\Pi_N$, the single term for each entry, the pairwise term between any two entries and cumulative terms along either multiple rows or multiple columns are defined
with respect to the positive dataset \( D^+ \), the negative dataset \( D^- \) and the whole training dataset \( D \) respectively as follows.

**Single terms.** For each entry \( \Pi_N(n,k) \), denote the set of positive data and negative data which fall into it respectively by,

\[
D^+(n,k) = \{ x : g(x,n) \in \Pi_N(n,k), x \in D^+ \}
\]

\[
D^-(n,k) = \{ x : g(x,n) \in \Pi_N(n,k), x \in D^- \}
\]

and denote their individual cardinalities by,

\[
\Omega^+(n,k) = |D^+(n,k)| \quad \text{and} \quad \Omega^-(n,k) = |D^-(n,k)|
\]

Then, the total training data in the entry \( \Pi_N(n,k) \) is defined by \( D(n,k) = D^+(n,k) \cup D^-(n,k) \) and its cardinality is \( \Omega(n,k) = \Omega^+(n,k) + \Omega^-(n,k) \).

Consider the left side of an entry \( \Pi_N(n,k) \) exclusive, denote the set of positive training data which fall into it and its cardinality respectively by,

\[
D^+_l(n,k) = \bigcup_{i=1}^{k-1} D^+(n,i)
\]

\[
\Omega^+_l(n,k) = |D^+_l(n,k)| = \sum_{i=1}^{k-1} \Omega^+(n,i)
\]

and the set of positive training data which fall into the right side of a entry \( \Pi_N(n,k) \) exclusive and its cardinality respectively by,

\[
D^+_r(n,k) = \bigcup_{i=k+1}^{K} D^+(n,i)
\]

\[
\Omega^+_r(n,k) = |D^+_r(n,k)| = \sum_{i=k+1}^{K} \Omega^+(n,i)
\]

Similarly, \( D^-_l(n,k) \), \( \Omega^-_l(n,k) \), \( D^-_r(n,k) \) and \( \Omega^-_r(n,k) \) are defined for negative data and then for the total training data, \( D_l(n,k) = D^+_l(n,k) \cup D^-_l(n,k) \) and \( \Omega_l(n,k) = \Omega^+_l(n,k) + \Omega^-_l(n,k) \), and \( D_r(n,k) = D^+_r(n,k) \cup D^-_r(n,k) \) and \( \Omega_r(n,k) = \Omega^+_r(n,k) + \Omega^-_r(n,k) \).
**Pairwise terms.** For any two entries $\Pi_N(n, k)$ and $\Pi_N(n', k')$, denote the set of positive training data which appear in both entries and its cardinality respectively by,

\[ D^+(n, k; n', k') = \{ x; g(x, n) \in \Pi_N(n, k) \text{ and } g(n', k') \in \Pi_N(n', k'), x \in D^+ \} \]

\[ = D^+(n, k) \cap D^+(n', k') \]

\[ \Omega^+(n, k; n', k') = |D^+(n, k; n', k')| \]

In the same way, $D^-(n, k; n', k')$ and $\Omega^-(n, k; n', k')$ are defined for negative data and $D(n, k; n', k') = D^+(n, k; n', k') \cup D^-(n, k; n', k')$ and $\Omega(n, k; n', k') = \Omega^+(n, k; n', k') + \Omega^-(n, k; n', k')$ are defined with respect to the whole training dataset $D$. Furthermore, by definition, all the pairwise terms are symmetric, e.g., $D^+(n, k; n', k') = D^+(n', k'; n, k)$ and $\Omega^+(n, k; n', k') = \Omega^+(n', k'; n, k)$.

Similarly, denote the set of positive training data which appear in both the left side of an entry $\Pi_N(n, k)$ and the left side of another entry $\Pi_N(n', k')$ and its cardinality respectively by,

\[ D^l_{l \rightarrow l}(n, k; n', k') = D^l_1(n, k) \cap D^l_1(n', k') \]

\[ \Omega^l_{l \rightarrow l}(n, k; n', k') = |D^l_{l \rightarrow l}(n, k; n', k')| \]

and the set of positive training data which appear in both the left side of $\Pi_N(n, k)$ and the right side of $\Pi_N(n', k')$ and it cardinality respectively by,

\[ D^l_{l \rightarrow r}(n, k; n', k') = D^l_1(n, k) \cap D^r_1(n', k') \]

\[ \Omega^l_{l \rightarrow r}(n, k; n', k') = |D^l_{l \rightarrow r}(n, k; n', k')| \]

In the same way, the pairwise terms of positive training data from the right side of $\Pi_N(n, k)$ to the left side of $\Pi_N(n', k')$, and to the right side of $\Pi_N(n', k')$ are
defined respectively by,

\[ D_{r \to l}^+(n, k; n', k') = D_r^+(n, k) \cap D_l^+(n', k') \]
\[ \Omega_{r \to l}^+(n, k; n', k') = |D_{r \to l}^+(n, k; n', k')| \]
\[ D_{r \to r}^+(n, k; n', k') = D_r^+(n, k) \cap D_r^+(n', k') \]
\[ \Omega_{r \to r}^+(n, k; n', k') = |D_{r \to r}^+(n, k; n', k')| \]

By definition, \( D_{l \to r}(n, k; n', k') = D_{r \to l}(n', k'; n, k) \) and then \( \Omega_{l \to r}(n, k; n', k') = \Omega_{r \to l}(n', k'; n, k) \).

Furthermore, the four pairwise terms with respect to negative data \( D^- \) and the whole training data \( D \) are defined respectively.

**Cumulative pairwise terms.** Consider all the left sides of entries \( \{(i, k_i); i = 1 \cdots n\} \), denote the set of positive training data which are double-counted in \( D_l^+(n, k_n) \) with respect to the others \( D_l^+(i, k_i) \)'s \( (i = 1, \cdots, n - 1) \) by,

\[ D_{l \to l}^+(n, \{k_i\}_{i=1}^n) = D_l^+(n, k_n) \cap (\cup_{i=1}^{n-1} D_l^+(i, k_i)) \tag{5.5} \]

with the cardinality denoted by \( \Omega_{l \to l}^+(n, \{k_i\}_{i=1}^n) = |D_{l \to l}^+(n, \{k_i\}_{i=1}^n)| \).

And, the set of double-counted training data from the left to the right, from the right to the left and from the right to the right are defined respectively by,

\[ D_{l \to r}^+(n, \{k_i\}_{i=1}^n) = D_r^+(n, k_n) \cap (\cup_{i=1}^{n-1} D_l^+(i, k_i)) \tag{5.6} \]
\[ D_{r \to l}^+(n, \{k_i\}_{i=1}^n) = D_l^+(n, k_n) \cap (\cup_{i=1}^{n-1} D_r^+(i, k_i)) \tag{5.7} \]
\[ D_{r \to r}^+(n, \{k_i\}_{i=1}^n) = D_r^+(n, k_n) \cap (\cup_{i=1}^{n-1} D_r^+(i, k_i)) \tag{5.8} \]

as well as \( \Omega_{l \to r}^+(n, \{k_i\}_{i=1}^n), \Omega_{r \to l}^+(n, \{k_i\}_{i=1}^n) \) and \( \Omega_{r \to r}^+(n, \{k_i\}_{i=1}^n) \). In the same way, the four cumulative pairwise terms with respect to negative data set \( D^- \) and the whole training dataset \( D \) are defined.
Figure 5.4: Illustration of selecting the allowable bounds on accuracy performance for a learnt computing process.

5.2.4 The Risk Function of a Decision Policy

The objective of a decision policy is to minimize the expected number of total tested terms $n$’s in the to-arrive scores (i.e., expected total computing cost), denoted by $E[\text{Cost}(\Pi_N)]$, given allowable accuracy performance bounds (i.e., FPR, denoted by $a$, and FNR, denoted by $b$). It is equivalent to solve the following constrained optimization problem,

$$\min_{\Pi_N} E[\text{Cost}(\Pi_N)]$$

subject to:

$p(\text{FP}; \Pi_N) \leq a$

$p(\text{FN}; \Pi_N) \leq b$

where $(a, b)$ are often specified according to vision goals and often selected based on the ROC of $f(x)$, e.g., the shadow area under the ROC as illustrated in Fig. 5.4. reducing $a$ leads to increase $b$, and vice versa. The risk function of a decision policy is defined after $E[n; \Pi_N]$, $p(\text{FP}; \Pi_N)$ and $p(\text{FN}; \Pi_N)$ are derived as follows.
The expected total computing cost using $\Pi_N$ is defined by,

$$ E[\text{Cost}(\Pi_N)] = \sum_{n=1}^{N} n \times [ (\Omega_l(n, \tau_{n,1}) + \Omega_r(n, \tau_{n,2})) - \Omega_{l\rightarrow l}(n, \{\tau_{i,1}\}_{i=1}^{n}) - \Omega_{l\rightarrow r}(n, \{\tau_{i,1}\}_{i=1}^{n-1}, \tau_{n,2}) + \Omega_{r\rightarrow l}(n, \{\tau_{i,2}\}_{i=1}^{n-1}, \tau_{n,1}) - \Omega_{r\rightarrow r}(n, \{\tau_{i,2}\}_{i=1}^{n})] $$

(5.10)

where in the $[\cdot]$, the first two terms are single terms and the other four terms are the cumulative pairwise terms which represent double-counted data among the single terms. Then, the expected computing cost for a random example using $\Pi_N$ is defined by,

$$ E[n; \Pi_N] = \sum_{n=1}^{N} n \times p(n; \Pi_N) = \frac{1}{|D|} E[\text{Cost}(\Pi_N)] $$

(5.11)

The probability of a FP in terms of $\Pi_N$ is defined by,

$$ p(\text{FP}; \Pi_N) = p(\Pi_N \text{ accept } x | x \in D^-) = p(k_1 > \tau_{1,2} | x \in D^-) + \sum_{n=2}^{N} p(\{\tau_{i,1} \leq k_i \leq \tau_{i,2}\}_{i=1}^{n-1}, k_n > \tau_{n,2} | x \in D^-) $$

$$ = \frac{\Omega_-(1, \tau_{1,2})}{|D^-|} + \sum_{n=2}^{N} \frac{\Omega_-(n, \tau_{n,2}) - \Omega_{l\rightarrow r}(n, \{\tau_{i,1}\}_{i=1}^{n-1}, \tau_{n,2}) - \Omega_{r\rightarrow r}(n, \{\tau_{i,2}\}_{i=1}^{n})}{|D^-|} $$

(5.12)

$$ = \sum_{n=1}^{N} \frac{\Omega_-(n, \tau_{n,2}) - \Omega_{l\rightarrow r}(n, \{\tau_{i,1}\}_{i=1}^{n-1}, \tau_{n,2}) - \Omega_{r\rightarrow r}(n, \{\tau_{i,2}\}_{i=1}^{n})}{|D^-|} $$

where by definition $\Omega_{l\rightarrow r}(1, \tau_{1,2}) = \Omega_{r\rightarrow r}(1, \tau_{1,2}) = 0$.

The expected number of FPs in terms of $\Pi_N$ is then defined by,

$$ E[\#\text{FP}; \Pi_N] = p(\text{FP}; \Pi_N) \times |D^-| $$

(5.13)

$$ = \sum_{n=1}^{N} [\Omega_-(n, \tau_{n,2}) - \Omega_{l\rightarrow r}(n, \{\tau_{i,1}\}_{i=1}^{n-1}, \tau_{n,2}) - \Omega_{r\rightarrow r}(n, \{\tau_{i,2}\}_{i=1}^{n})] $$
Similarly, the probability of a FN in terms of $\Pi_N$ is defined by,

\[
p(FN; \Pi_N) = p(\Pi_N \text{ reject } x \mid x \in D^+) \tag{5.14}
\]

\[
= p(k_1 < \tau_{1,1} \mid x \in D^-) + \sum_{n=2}^{N} p(\{\tau_{i,1} \leq k_i \leq \tau_{i,2}\}_{i=1}^{n-1}, k_n < \tau_{n,1} \mid x \in D^+)
\]

\[
= \frac{\Omega_t^+(1, \tau_{1,1})}{|D^+|} + \sum_{n=2}^{N} \frac{\Omega_t^+(n, \tau_{n,1}) - \Omega_{t \rightarrow t}^+(n, \{\tau_{i,1}\}_{i=1}^{n}) - \Omega_{t \rightarrow t}^+(n, \{\tau_{i,2}\}_{i=1}^{n-1}, \tau_{n,1})}{|D^+|}
\]

where by definition $\Omega_{t \rightarrow t}^+(1, \tau_{1,1}) = \Omega_{t \rightarrow t}^+(1, \tau_{1,1}) = 0$.

The expected number of FNs in terms of $\Pi_N$ is correspondingly defined by,

\[
E[\#FN; \Pi_N] = p(FN; \Pi_N) \times |D^+| \tag{5.15}
\]

\[
= \sum_{n=1}^{N} [\Omega_t^+(n, \tau_{n,1}) - \Omega_{t \rightarrow t}^+(n, \{\tau_{i,1}\}_{i=1}^{n}) - \Omega_{t \rightarrow t}^+(n, \{\tau_{i,2}\}_{i=1}^{n-1}, \tau_{n,1})]
\]

In object detection, giving bounds $(a, b)$ on FPR and FNR is equivalent to specify a pair loss costs: the FP loss cost, denoted by $C_{FP}$, and the FN loss cost, denoted by $C_{FN}$. So, learning the decision policy naturally addresses the loss cost-sensitive object detection. This connection is built by defining the risk function of a decision policy.

**The risk function of $\Pi_N$.** The risk function is defined by,

\[
\text{Risk}(\Pi_N, C_{FP}, C_{FN}) = \frac{1}{|D|} \times \{E[\text{Cost}(\Pi_N)] + E[\#FP; \Pi_N] \cdot C_{FP} + E[\#FN; \Pi_N] \cdot C_{FN}\} \tag{5.16}
\]

which takes into account the computational cost, the FP and FN loss cost.
By using Eqn. 5.11, Eqn. 5.13 and Eqn. 5.15, Eqn. 5.16 is written as,

\[
\text{Risk}(\Pi_N, C_{FP}, C_{FN}) = E[n; \Pi_N] + p(\text{FP}; \Pi_N) \cdot \frac{|D^-|}{|D|} \cdot C_{FP} + p(\text{FN}; \Pi_N) \cdot \frac{|D^+|}{|D|} \cdot C_{FN} \tag{5.17}
\]

Then, the optimal decision policy \( \Pi_N^* \) (for notational simplicity, \( \Pi_N^* \) is used below) is defined by,

\[
\Pi_N^* = \arg \min_{\Pi_N} \text{Risk}(\Pi_N, C_{FP}, C_{FN}) \tag{5.18}
\]

The equivalence between Eqn. 5.9 and Eqn. 5.16 is proved as follows.

**Lemma.** The solution \( \Pi_N^* \) (Eqn. 5.18) is also the solution to the original constrained optimization problem (Eqn. 5.9) with \( a = p(\text{FP}; \Pi_N^*) \) and \( b = p(\text{FN}; \Pi_N^*) \).

**Proof.** Suppose there was a decision policy \( \Pi_N' \) solving Eqn. 5.9 with the smaller or equal computational cost \( E[n; \Pi_N'] \leq E[n; \Pi_N^*] \) and better or equal accuracy performance \( a' = p(\text{FP}; \Pi_N') \leq p(\text{FP}; \Pi_N^*) \) and \( b' = p(\text{FN}; \Pi_N') \leq p(\text{FN}; \Pi_N^*) \).

Since \( \Pi_N^* \) is optimal, so,

\[
\text{Risk}(\Pi_N^*, C_{FP}, C_{FN}) \leq \text{Risk}(\Pi_N', C_{FP}, C_{FN}) \tag{5.19}
\]

where,

\[
\text{LHS} = E[n; \Pi_N^*] + p(\text{FP}; \Pi_N^*) \cdot \frac{|D^-|}{|D|} \cdot C_{FP} + p(\text{FN}; \Pi_N^*) \cdot \frac{|D^+|}{|D|} \cdot C_{FN} \\
\leq \text{RHS} = E[n; \Pi_N'] + p(\text{FP}; \Pi_N') \cdot \frac{|D^-|}{|D|} \cdot C_{FP} + p(\text{FN}; \Pi_N') \cdot \frac{|D^+|}{|D|} \cdot C_{FN}
\]
Using the assumptions \( p(\text{FP}; \Pi^*_N) \leq p(\text{FP}; \Pi'_N) \) and \( p(\text{FN}; \Pi^*_N) \leq p(\text{FN}; \Pi'_N) \),

\[
\leq E[n; \Pi'_N] + p(\text{FP}; \Pi^*_N) \cdot \frac{|D^-|}{|D|} \cdot C_{FP} + p(\text{FN}; \Pi^*_N) \cdot \frac{|D^+|}{|D|} \cdot C_{FN}
\]

So \( E[n; \Pi^*_N] \leq E[n; \Pi'_N] \) which is conflicted with the assumption \( E[n; \Pi'_N] \leq E[n; \Pi^*_N] \). Then, by using backward induction, it proves that \( \Pi^*_N \) is the solution to the original constrained optimization problem (Eqn. 5.9) with \( a = p(\text{FP}; \Pi^*_N) \) and \( b = p(\text{FN}; \Pi^*_N) \).

Next, the risk function (Eqn. 5.16) is solved by dynamic programming. Then, through searching \((C_{FP}, C_{FN})\), the original constrained optimization problem (Eqn. 5.9) is solved with the near-optimal solution found.

### 5.3 Minimizing an Upper Bound of the Risk Function by DP

The risk function is unfolded as,

\[
\text{Risk}(\Pi_N, C_{FP}, C_{FN})
\]

\[
= \frac{1}{|D|} \cdot \left\{ E[\text{Cost}(\Pi_N)] + E[\#\text{FP}; \Pi_N] \cdot C_{FP} + E[\#\text{FN}; \Pi_N] \cdot C_{FN} \right\}
\]

\[
= \frac{1}{|D|} \cdot \sum_{n=1}^{N} n \cdot \left[ (\Omega_l(n, \tau_{n,1}) + \Omega_r(n, \tau_{n,2})) - \right.
\]

\[
(\Omega_{l \rightarrow r}(n, \{\tau_{i,1}\}_{i=1}^{n}) + \Omega_{l \rightarrow l}(n, \{\tau_{i,1}\}_{i=1}^{n-1}, \tau_{n,2}) + \Omega_{r \rightarrow l}(n, \{\tau_{i,2}\}_{i=1}^{n-1}, \tau_{n,1}) + \Omega_{r \rightarrow r}(n, \{\tau_{i,2}\}_{i=1}^{n})) +
\]

\[
\sum_{n=1}^{N} [\Omega^-_{l}(n, \tau_{n,2}) - \Omega^-_{l \rightarrow r}(n, \{\tau_{i,1}\}_{i=1}^{n-1}, \tau_{n,2}) - \Omega^-_{r \rightarrow l}(n, \{\tau_{i,2}\}_{i=1}^{n})] \cdot C_{FP} +
\]

\[
\sum_{n=1}^{N} [\Omega^+_{l}(n, \tau_{n,1}) - \Omega^+_{l \rightarrow l}(n, \{\tau_{i,1}\}_{i=1}^{n}) - \Omega^+_{r \rightarrow l}(n, \{\tau_{i,2}\}_{i=1}^{n-1}, \tau_{n,1})] \cdot C_{FN}
\]

\]
By re-arranging the single and cumulative pairwise terms, it is reproduced as,

\[
\text{Risk}(\Pi_N, C_{FP}, C_{FN}) \tag{5.20}
\]

\[
= \sum_{n=1}^{N} [\text{risk}(n, C_{FP}, C_{FN}) - \text{risk}(1 \leadsto n, C_{FP}, C_{FN})]
\]

which in general is NP-hard due to the non-Markov property in the cumulative pairwise term, e.g., \(\Omega_{l \rightarrow l}(n, \{\tau_i, 1\}_{i=1}^n)\) is decided by \(\{\tau_i, 1\}_{i=1}^n\) jointly. So, by relaxing the cumulative pairwise terms, an upper bound of the risk function is minimized instead as derived as follows.

First, the risk accounting for a single row in the decision policy is defined by,

\[
\text{risk}(n, C_{FP}, C_{FN}) \tag{5.21}
\]

\[
= \frac{1}{|D|} \cdot [n \cdot (\Omega_{l}(n, \tau_{n,1}) + \Omega_{r}(n, \tau_{n,2})) + \Omega_{r}^{-}(n, \tau_{n,2}) \cdot C_{FP} + \Omega_{l}^{-}(n, \tau_{n,1}) \cdot C_{FN}]
\]

where three DP tables are created, \(S_1(n, \tau_{n,1}, \tau_{n,2}) = n \cdot (\Omega_{l}(n, \tau_{n,1}) + \Omega_{r}(n, \tau_{n,2}))\), \(S_2(n, \cdot, \tau_{n,2}) = \Omega_{r}^{-}(n, \tau_{n,2})\) and \(S_3(n, \tau_{n,1}, \cdot) = \Omega_{l}^{-}(n, \tau_{n,1})\) ("\cdot" represents all entries). In practice, due to \(\tau_{n,1} \leq \tau_{n,2}\), the three DP tables are sparsely recorded to save memory. Then, the risk for single terms can be computed very fast when the FP and FN loss costs \((C_{FP} \text{ and } C_{FN})\) are changed during on-line computing.

Second, the risk accounting for cumulative pairwise terms is defined by,

\[
\text{risk}(1 \leadsto n, C_{FP}, C_{FN}) \tag{5.22}
\]

\[
= \frac{1}{|D|} \cdot \{n \cdot [(\Omega_{l \rightarrow l}(n, \{\tau_i, 1\}_{i=1}^n) + \Omega_{l \rightarrow r}(n, \{\tau_i, 1\}_{i=1}^{n-1}, \tau_{n,2}) +
\Omega_{r \rightarrow l}(n, \{\tau_{i,2}\}_{i=1}^{n-1}, \tau_{n,1}) + \Omega_{r \rightarrow r}(n, \{\tau_{i,2}\}_{i=1}^n)] +
\Omega_{l \rightarrow r}(n, \{\tau_{i,1}\}_{i=1}^{n-1}, \tau_{n,2}) + \Omega_{r \rightarrow r}(n, \{\tau_{i,2}\}_{i=1}^n)] \cdot C_{FP} +
\Omega_{l \rightarrow l}^{-}(n, \{\tau_{i,1}\}_{i=1}^n) + \Omega_{r \rightarrow l}^{-}(n, \{\tau_{i,2}\}_{i=1}^{n-1}, \tau_{n,1}) \cdot C_{FN}\}
\]

Recall that by definition \(D_{l \rightarrow l}^{-}(n, \{\tau_{i,1}\}_{i=1}^n) = D_{l}^{-}(n, \tau_{n,1}) \cap (\cup_{i=1}^{n-1} D_{l}^{-}(i, \tau_{i,1}))\)
and $D_l^-(i, \tau_{n-1,1}) \subseteq \bigcup_{i=1}^{n-1} D_l^-(i, \tau_{i,1})$. So,
\[ D_l^-(n, \{\tau_{i,1}\}_{i=n-1}^n) \subseteq D_l^-(n, \{\tau_{i,1}\}_{i=1}^n) \]
\[ \Omega_l^-(n, \{\tau_{i,1}\}_{i=n-1}^n) \leq \Omega_l^-(n, \{\tau_{i,1}\}_{i=1}^n) \]
and the same is for the other cumulative pairwise terms in $\text{risk}(1 \leadsto n, C_{FP}, C_{FN})$.

By using those relaxations, a lower bound of the risk for cumulative terms, denoted by $\text{risk}(n-1, n, C_{FP}, C_{FN}) \leq \text{risk}(1 \leadsto n, C_{FP}, C_{FN})$, is defined by,
\[
\text{risk}(n-1, n, C_{FP}, C_{FN}) = \frac{1}{|D|} \cdot \{ P_1(n, \tau_{n-1,1}, \tau_{n-1,2}, \tau_{n,1}, \tau_{n,2}) + P_2(n, \tau_{n-1,1}, \tau_{n-1,2}; \tau_{n,1};) \cdot C_{FP} + P_3(n, \tau_{n-1,1}, \tau_{n-1,2}, \tau_{n,1};) \cdot C_{FN} \}
\]
where similar to compute the risk for single terms, three sparse DP tables $P_i(\cdot)$ ($i = 1, 2, 3$) are calculated off-line.

Then, an upper bound of the risk function $\text{Risk}(\Pi_N, C_{FP}, C_{FN})$ (Eqn. 5.20) is obtained,
\[
\hat{\text{Risk}}(\Pi_N, C_{FP}, C_{FN}) = \sum_{n=1}^{N} [\text{risk}(n, C_{FP}, C_{FN}) - \text{risk}(n-1, n, C_{FP}, C_{FN})]
\]
which can be minimized by DP as follows.
The DP algorithm. Let $\Pi_{N-n} = \{\tau_1, \cdots, \tau_{N-n}\}$. Minimizing the risk function (Eqn. 5.24) is expressed by the recursion,

$$
\min_{\Pi_N} \hat{\text{Risk}}(\Pi_N, C_{FP}, C_{FN}) = \min_{\Pi_N} \sum_{n=1}^{N} [\text{risk}(n, C_{FP}, C_{FN}) - \text{risk}(n-1, C_{FP}, C_{FN})]
$$

$$
= \min_{\tau_N} \min_{\Pi_{N-1}} \sum_{n=1}^{N} [\text{risk}(n, C_{FP}, C_{FN}) - \text{risk}(n-1, C_{FP}, C_{FN})]
$$

$$
= \min_{\tau_N} \{\text{risk}(N, C_{FP}, C_{FN}) + \min_{\Pi_{N-1}} [\hat{\text{Risk}}(\Pi_{N-1}, C_{FP}, C_{FN}) - \text{risk}(N-1, C_{FP}, C_{FN})]\}
$$

5.4 Searching the Near-Optimal Decision Policy Given Allowable Accuracy Performance Bounds

Given a pair of allowable bounds $(a, b)$, the corresponding near-optimal decision policy (near-optimal due to the discretization in searching), denoted by $\Pi_N^*(a, b)$, is sought by searching $\Pi_N^*(C_{FP}, C_{FN})$ (which is solved by DP) with $C_{FP} \in [0, C_{FP}^{\max}]$ and $C_{FN} \in [0, C_{FN}^{\max}]$ until $p(FP; \Pi_N^*(C_{FP}, C_{FN}))$ and $p(FN; \Pi_N^*(C_{FP}, C_{FN}))$ are as close with the specified bounds $a$ and $b$ as possible. $C_{FP}^{\max}$ and $C_{FN}^{\max}$ are derived as follows.

Denote by $\Pi_N^0$ the “naive” decision policy where a decision is made only after the full $f(x) = g(x, N)$ is tested and with $\tau_{N,1} = \tau_{N,2} = \tau$ used (i.e., no early decisions are made). So, the corresponding risk is,

$$
\text{Risk}(\Pi_N^0, C_{FP}, C_{FN}) = N + \frac{\Omega^{-}(N, \tau)}{|D|} \cdot C_{FP} + \frac{\Omega^{+}(N, \tau)}{|D|} \cdot C_{FN}
$$

(5.26)

where $\Omega^{-}(N, \tau) = \Omega^{+}(N, \tau) = 0$ if $f(x)$ perfectly separates the positive and negative training data using the threshold $\tau$, or $\Omega^{-}(N, \tau)$ and $\Omega^{+}(N, \tau)$ take
small values with respect to $|D|$. Then, the risk $\text{Risk}(\Pi_{N}^{0}, C_{FP}, C_{FN}) \approx N$, i.e.,
the full computational cost.

By definition, $\text{Risk}(\Pi_{N}^{0}, C_{FP}, C_{FN})$ is greater or equal to the risk of $\Pi_{N}^{*}(C_{FP}, C_{FN})$,

$$N \geq \text{Risk}(\Pi_{N}^{*}, C_{FP}, C_{FN})$$

(5.27)

Then,

$$C_{FP} < \frac{N \cdot |D|}{a \cdot |D^{-}|} \triangleq C_{FP}^{\max}$$

(5.28)

Similarly, $C_{FN}^{\max} = \frac{N \cdot |D|}{b \cdot |D^{+}|}$. Then, denote by $(C_{FP}^{*}, C_{FN}^{*})$ the best pair of loss costs for
the given pair of accuracy performance bounds $(a, b)$.

5.5 The Risk of a Testing Example after Computed

Given the learnt near-optimal decision policy $\Pi_{N}^{*}(C_{FP}^{*}, C_{FN}^{*})$ and a testing example $x$, what are the risks for accepting $x$ and rejecting $x$ respectively after $x$ is actually computed in terms of $\Pi_{N}^{*}(C_{FP}^{*}, C_{FN}^{*})$? These will measure the quality of solution in cost-sensitive object detection/parsing. Denote by $n_{x}$ the number of actually tested terms for $x$, i.e., $1 \leq n_{x} \leq N$, and the to-arrive scores of $x$ are denoted by $G(x) = \{g(x, 1), \cdots, g(x, n_{x})\}$.

(1) The risk of accepting $x$, denoted by $\text{risk}(\text{Acpt}|x)$, is defined by,

$$\text{risk}(\text{Acpt}|x) = n_{x} + p(\text{FP}|G(x)) \times C_{FP}^{*}$$

(5.29)

where by definition $\tau_{n,1} \leq g(x, n) \leq \tau_{n,2} (\forall n \in [1, n_{x}-1])$ and $g(x, n_{x}) > \tau_{n_{x},2}$
and $p(\text{FP}|G(x))$ is calculated based on the actual to-arrive scores of $x$,

$$p(\text{FP}|G(x)) = \frac{\Omega_{\tau}^{-}(n_{x}, g(x, n_{x})) - \Omega_{\tau_{1} \rightarrow \tau_{2}}^{-}(n_{x}, \{\tau_{i,1}\}_{i=1}^{n_{x}-1}, g(x, n_{x})) - \Omega_{\tau_{2} \rightarrow \tau_{2}}^{-}(n_{x}, \{\tau_{i,2}\}_{i=1}^{n_{x}-1}, g(x, n_{x}))}{|D^{-}|}$$

(5.30)
(2) The risk of rejecting $x$, denoted by $\text{risk}(R\text{jct}|x)$, is defined by,

$$\text{risk}(R\text{jct}|x) = n_x + p(\text{FN}|G(x)) \times C_{FN}^*$$

(5.31)

where also by definition $g(x, n) < \tau_{n,1} (\forall n \in [1, n_x])$ and similarly $p(\text{FN}|G(x))$ is calculated by,

$$p(\text{FN}|G(x)) = \frac{\Omega^+_i(n_x, g(x, n_x)) - \Omega^+_{i-1}(n_x, \{\tau_{i,1}\}_{i=1}^{n_x-1}, g(x, n_x)) - \Omega^+_{r-1}(n_x, \{\tau_{i,2}\}_{i=1}^{n_x-1}, g(x, n_x))}{|D^+|}$$

(5.32)

5.6 Experiments

5.6.1 Learning the decision policy for a terminal node

In this experiments, the Adaboost classifier [Viola and Jones, 2004] is learnt for $16 \times 16$ human faces (as $\alpha$ process) consisting of 250 boosted classifiers. The training dataset includes $|D^+| = 9471$ positive images and $|D^-| = 36285$ negative images. The boosted 250 weak classifiers are factorized into 10 subsets (i.e., $f_i(x)$, $i = 1, \cdots, 10$) each of which consists of 25 weak classifiers, and the orders of weak classifiers in $f_i$’s are the same as the boosted orders in learning the Adaboost classifier. The AUC of the ROC for learning is 0.99957. Based on the ROC, a pair of FPR and FNR, $a = 0.0099$ and $b = 0.0120$, is specified, and then $C^{\text{max}}_{FP}$ and $C^{\text{max}}_{FN}$ are computed based on Eqn. 5.28, as shown in the middle plot of Fig. 5.5. To illustrate the learnt decision policies for different combinations of the FP and FN costs, four special situations are shown in Fig. 5.5. In the left-bottom figure, since $C_{FP} = C_{FN} = 0$, the decision policy then does not start computing due to no loss costs at all. In the right-bottom figure, $C_{FP} = C^{\text{max}}_{FP}$ and $C_{FN} = 0$, the decision policy computes $g(x, 1)$ and then let $\tau_{1,1}$ take the maximal score of negatives (so that no FPs occur, but do not care FNs). Similarly, in the left-top
Figure 5.5: This figure shows learnt decision policies for the $\alpha$ process of human faces using Adaboost classifier. The four decision policies correspond to four pairs of FP and FN loss costs (see the four corners in the middle figure). See texts for the explanations of the four learnt decision policies.
Figure 5.6: This figure shows another example of the learnt decision policies for human face Adaboost classifier with corresponding FPs and FNs shown, where the first column shows the average image. One interesting observation is that for the FPs the average image looks similar to the one of TPs when \( n \approx N \).

The decision policy also computes \( g(x, 1) \) and the let \( \tau_{1,2} \) take the minimal score of positives (so that no FNs happen). In the right-top figure, both \( C_{FP} \) and \( C_{FN} \) take the maximal values and \( C_{FN} > C_{FP} \), the decision policy then allow earlier acceptance then rejections. Fig. 5.6 shows the learnt decision policy for another pair of loss FP and FN costs (the yellow points in the right-top figure) with corresponding FNs and FPs shown (the first column shows the average image). One interesting observation is that for the FPs the average image looks similar to the one of TPs when \( n \approx N \).
Figure 5.7: This figure shows four learnt decision policy for the deformable part model of an And-node using pedestrian as an example with four pairs of FP and FN loss cost. See texts for the explanations of the four learnt decision policies.
Figure 5.8: Another example of the learnt decision policy for pedestrian deformable part model.

5.6.2 Learning the decision policy for an And-node

In this experiment, the deformable part model [Felzenszwalb et al., 2010] with one root node and 8 part nodes is learned for frontal view person in VOC 2007 dataset. The training dataset consists of $|D^+| = 2560$ positive images and $|D^-| = 15793$ negative images. For deformable part model, $f_1()$ represents the root node and each $f_2(i) \ (i = 2, \cdots, 9)$ represents a part node. Fig. 5.7 shows the learnt decision policies for the four corners when searching the loss FP and FN costs and shares the similar interpretations with the figures in Fig. 5.8.
CHAPTER 6

Scheduling BU/TD Processes by Best-First Heuristic Search

This chapter presents a theoretical study of scheduling BU/TD computing processes in an AoG, aiming at two objectives: (i) Advancing on-line computational efficiency given allowable bounds on accuracy performance; (ii) Adapting the computing orders to different vision tasks. The decision policy learnt in Chapter 5 is context free in the sense that only the inside information for a terminal node or an And-node (i.e., the scoring function $f(x)$) is used. In this chapter, the decision policy is extended to context-sensitive one by incorporating information outside $f(x)$ in the parse graph. Then, the scheduling problem is formulated under the best-first heuristic search [Pearl, 1984] and a agenda-chart scheduling algorithm is presented. The scheduling space is defined as the product space of the $\alpha$-$\beta$-$\gamma$ computing process space and the pose space (i.e., location, scale and orientation). Then, the risk function is defined on configurations of the scheduling space to be minimized by the agenda-chart scheduling algorithm. Concretely, the risk function factorizes into two parts: to-arrive risk in the chart and to-go risk in the agenda. The to-arrive risk further consists of two terms: the inside risk computed from the tested and accepted elements (then FPs may occur) while the outside risk from the tested but rejected elements (then FNs may happen). The to-go risk is estimated for elements which can be predicated by the chart and are
either tested but rejected in previous steps or not tested yet. The to-go elements are then prioritized according to the estimated amount of decreased risk if tested. The most promising to-go elements are then selected to be actually explored, and then the agenda and chart are updated accordingly.

6.1 Background

It has long been recognized that both bottom-up and top-down inference processes contribute to different vision tasks at all the low, middle and high vision levels [Riesenhuber and Poggio, 1999; Thorpe et al., 1996; Borenstein and Ullman, 2008; Levin and Weiss, 2009]. Although many efforts have been dedicated to develop a framework to combine them in a robust and efficient way [Ullman, 1984; Tu et al., 2005; Han and Zhu, 2009], the scheduling problem of bottom-up and top-down inference processes is still open in the vision community.

Psychological Studies In his seminal work [Ullman, 1984], Ullman proposed the paradigm, “visual routines”, that for different high-level vision tasks, there exists a specialized sequence of operators selected from a set of reusable elemental operators. His focus was mainly based on the cognitive perspective, and a set of operators are listed: shifting the processing focus, indexing a salient item for further processing, spreading activation over an area delimited by boundaries, tracing boundaries, and marking a location or object for future reference. Following the idea of visual routines, a few works are addressed such as visual attention [Rao, 1998]. In active vision and visual attention, people are interested in the underlying mechanism guiding “where to look next” and “what to fixate”, and it is the common recognition that they are driven by both bottom-up and top-down inference processes.
Manual Scheduling. In the 1990s, in order to reuse different vision procedures such as image smoothing, edge and corner detection, there are some computer vision software environments in which application programmer can interactively schedule and specify a set of vision procedures from a predefined library by using a graphical user interface (GUI). For example, Khoros [Rasure and Kubica, 1994] allowed programmer to select procedures from menus and graphically connect the output of one procedure to the input of another. CVIPtools [Umbaugh, 1997] works in a similar way but primarily designed for academic use. The Image Understanding Environment (IUE) is primarily an object-oriented software library, but also includes a GUI for sequencing procedures [Mundy, 1995]. These can be looked as some interactive implementations of the idea of visual routines. Beyond these interactive design, a few works model object recognition under Markov decision process (MDP) framework [Draper et al., 2000] with some insights obtained.

Bayesian Maximum A Posterior (MAP) Inference. In hierarchical models, there are two basic types of inference processes, bottom-up and top-down, and they are often used with three kinds of strategies: (i) Pure bottom-up inference which passes messages in a feed-forward manner in the hierarchy, starting from data-driven features [Riesenhuber and Poggio, 1999; Serre et al., 2007; Aycinena et al., 2008; Zhu et al., 2010]; (ii) Pure top-down inference which passes messages in a feed-back manner in the hierarchy, starting from template matching [Yuille et al., 1992; Todorovic and Ahuja, 2008a; Demirci et al., 2009]; and (iii) One pass of bottom-up inference followed by one phase of top-down inference [Tu et al., 2005; Zhu et al., 2010; Epshtein et al., 2008; Borenstein and Ullman, 2008; Levin and Weiss, 2009; Demirci et al., 2006]. The three kinds of computing strategies did not address the scheduling problem. On the other hand, some recent human vision experiments show that humans can detect scene and object categories as
fast as the low level image primitives and the human visual system schedules the computing in an very effective way [Thorpe et al., 1996] (but how the human visual system handles those is still unclear to vision researchers).

**Empirical Risk Minimization (ERM) in Machine Learning.** There are three types of strategies: (i) *Cascade in linear model.* To overcome the computationally expensive cost of the sliding-window method, there are some improvements proposed in recent literature. During online computing, most of sliding windows belong to background, which can be ruled out with some simple and computationally cheap testing. Based on this assumption, cascade boosting methods [Viola and Jones, 2004] train a multi-stage classifier in which computationally expensive testing are only applied on those windows survived so that online computing time is reduced. (ii) *Coarse-to-fine in tree structure.* [Blanchard and Geman, 2005; Amit et al., 2004] organize all the testing in a hierarchy such as decision tree based on a top-down coarse-to-fine partition of the pose space of the object to be detected, and from the root node to each leaf node, each testing is selected in terms of keeping the false negative equal to zero. In essence, both approaches work in an attentional manner, shifting computational expensive testing onto those windows survived from earlier simple testing dedicated for backgrounds so more promising for targets appearing. (iii) *Joint boosting and boosted tree method to explore sharing.* The two kinds of methods stated above are mainly focus on how to efficient detect a single object represented by a classifier implicitly or a hierarchy with only leaf nodes linking to image data explicitly. In order to simultaneously detect multi-objects, joint boosting [Torralba et al., 2007] sequentially selects weak classifiers in terms of maximally minimizing the joint classification error at each step. Boosted tree methods [Tu, 2005] boost a tree classifier in which samples are probabilistically and sequentially divided
based on the output of each node of the tree. In spirits, these methods can be
looked as coarse-to-fine strategies for among multi-objects.

**Algorithmic Optimization.** Instead of working in a sliding window way, effi-
cient search algorithms based on branch-and-bound directly search sub-windows
of object instances through bounding the quality of a testing function such as a
trained classifier from the upper and the lower for any given sub-window. The
key is to create the lower and upper bounds, and current approaches [Lampert
et al., 2009] only adapt well for classifiers trained in terms of bag-of-word based
features. For those hierarchical models with only leaf nodes linking to image data,
dynamic programming [Rabiner, 1989] or A* search [Pearl, 1984] is often used to
do the inference such as horse and human body parsing [Zhu et al., 2010] and
long curve tracing [Felzenszwalb and McAllester, 2007] and car parsing [Kokkinos
and Yuille, 2009] and some others [Moreels et al., 2004]. Dynamic programming
is often used together with a pruning stage handling the combinatorial explosion.
A* based algorithms mainly use a certain abstraction of the representation of the
object at a coarse level as the lower bound for cost-to-go, and then prioritize all
the already explored sub-windows.

**Limitations of Existing Work** Despite the remarkable progresses stated
above, there are two limitations in existing work.

1. The inference is usually formulated with the assumption of zero-one loss, and
   further assume there was one globally optimal solution. Thus, it does not
   take into account for the loss costs (utilities) and different vision goals. As
   mentioned above, however, visual inference can be looked as an infinite goal-
   guided computing process. So, the inference should be active in the sense
   that it can adapt to different vision tasks and datasets.
(2) The objective function of inference does not take into account the computational cost explicitly (except for a theoretical study in [Blanchard and Geman, 2005] and a case study in astronomy projects [Meinshausen et al., 2009], but both of them do not consider the loss costs). Actually, limited computing resource is the main cause for the visual attention mechanism in the primate visual system.

6.2 Overview of Goal-Guided Scheduling

Recall that when BU/TD computing processes are integrated for object parsing using AoG formulated under Bayesian framework in Chapter 4, the goal is to maximize a posterior probability with zero-one loss cost used for each And-node. So, all BU/TD computing processes are first actually and fully computed (i.e., full exploration) and then parse graphs are constructed by BFS (i.e., full exploitation). In computational and loss cost-sensitive object detection using AoG, for each And-node \( A \), either a pair of loss costs (FP loss cost \( C_{FP}^A \) and FN loss cost \( C_{FN}^A \)) or a pair of allowable upper bounds on accuracy performance (FPR \( a_A \) and FNR \( b_A \)) is specified to account for different vision goals, and then the goal is to minimize the risk function which takes into account both the computing and loss costs to achieve the two objectives stated above. Starting with \( W_0 = \emptyset \) (Eqn.4.6), two questions arise in cost-sensitive object detection using AoG:

(1) Should the computing start for an input image \( I_A \), and if yes, which node and what process of the node should be tested first and where should the computing process run in the image lattice \( \Lambda \)?

(2) What computing order of nodes in an AoG should be used? Concretely, which node and what computing process should be tested next, and where,
given the outputs from computing the first node, and so on?

To address the two questions, heuristics of computing processes need to be exploited to compare the expected computing costs and corresponding loss costs (the two terms in the risk function) and then decisions could be made in terms of vision goals. For an And graph (consisting of terminal nodes and And-nodes), the first question can be handled in terms of the learnt decision policy (see Sec. 6.3). The decision policy learnt in Chapter 5 is context free in the sense that only inside information of the scoring function $f(x)$ is used without consider the outside information in the parse graph. To tackle the second question, the decision policy is extended to context-sensitive one to incorporate the information outside $f(x)$. By incorporating outside information in deriving the context-sensitive decision policy, the ratio between the population of positive and negative data is changed since the whole population is now conditioned on the given outside information. Then, the context-sensitive decision policy is used as heuristics guiding the scheduling in the scheduling space. The scheduling space is defined as the product space of the computing processes space from the AoG and the pose space from the testing image (i.e., all the locations, scales and orientations). The scheduling space is managed by an agenda and a chart during on-line computing, and then the scheduling algorithm uses the heuristics to explore the scheduling space and update the agenda-chart. When all computing costs are paid, the solution in the chart will approximately recover that obtained through integrating the $\alpha$-$\beta$-$\gamma$ processes by BFS in Chapter 4. One advantage of the scheduling algorithm proposed in this chapter is that the algorithm explicitly considers the vision-goal-guided trade-off between the accuracy performance and computational efficiency so that the algorithm can be terminated at any time but with the most decidable solution obtained given the computing cost.
6.3 Heuristics of the $\alpha$ Computing Process Using the Learnt Near-Optimal Decision Policy

To tackle the first question stated above, heuristics of the $\alpha$ computing processes of different nodes in an AoG are entailed, which use the learnt near-optimal decision policies.

Let $\Lambda$ be the image lattice and $I_\Lambda$ an input image. In cost-sensitive object detection with AoG, each And-node $A$ in an AoG can be placed through a terminal-node at different locations in $\Lambda$ at different scales (and even orientations which are not addressed here). In practice, an image pyramid $I = (I_{\Lambda_0}, \cdots, I_{\Lambda_L})$ is used to handle the scale space (where $\Lambda_0 = \Lambda$). Denote the pose space (i.e., each element corresponds to a sliding window in detection) by

$$\Phi = \{(l, x, y) : 0 \leq l \leq L, (x, y) \in \Lambda_l\}$$  \hspace{2cm} (6.1)

Furthermore, for each And-node $A$, the unknown number $n_A$ of instances of node $A$ occurring in $I_\Lambda$ is assumed to follow a Poison distribution with parameter $\lambda_A = \frac{1}{|A|} |\Lambda|$,

$$p(n_A = n) = \lambda_A^n e^{-\lambda_A} / n!$$  \hspace{2cm} (6.2)

where $|A|$ is the expected size of instances of node $A$ (e.g., $100 \times 100$ pixels).

So, if the $\alpha$ computing process of an And-node $A$ want to be started, the expected computing cost will be $|\Phi| \times E[n; \Pi^*_N(C^A_{FP}, C^A_{FN})]$, and the expected FP and FN loss costs will be $\sum_{n=0}^{+\infty} n \times p(n_A = n) \times p(FP; \Pi^*_N(C^A_{FP}, C^A_{FN})) \times C^A_{FP}$, and $\sum_{n=0}^{+\infty} (|\Phi| - n) \times p(n_A = n) \times p(FN; \Pi^*_N(C^A_{FP}, C^A_{FN})) \times C^A_{FN}$ respectively. By comparing the expected computing cost and corresponding loss costs, whether the computing should be started and which node should be tested first can then be decided in terms of the utility function specified by vision goals.
6.4 Learning Context-Sensitive Decision Policy

6.4.1 Factorizing the Scoring Function of α Process within a Parse Graph

For a parse graph $pg$, let $pg^{(1)}$ denote the parse graph where all nodes are computed and $pg^{(0)}$ the remaining parts for nodes not tested yet. Without loss of generality, consider an And-node $A$ in the remaining parts of a parse graph, i.e., $A \in V_{And}^{pg(0)}$, denote by $\omega^{(out)}$ the scores of $pg^{(1)}$, and still denote by $f(x)$ the scoring function of the And-node $A$. When learning the decision policy for the $\alpha$ process of And-node $A$ in Chapter 5, information from $pg^{(1)}$ is not exploited, i.e., assume $\omega^{(out)} = 0$, and then $f(x) \triangleq f(x|\omega^{(out)} = 0)$ is investigated individually and independently. To explicitly incorporate information outside $f(x)$, denote by $f(x|\omega^{(out)})$ the scoring function of And-node $A$ given the scores of nodes in $pg^{(1)}$ (i.e., $\omega^{(out)}$). Concretely, consider the integrated scoring function of a parse graph in Eqn.4.12 and the corresponding scoring components of the $\alpha$-$\beta$-$\gamma$ processes (Eqn. 4.13, Eqn. 4.14 and Eqn. 4.15 respectively), the scores of $\beta$-$\gamma$ processes are treated as the “outside” information, i.e., $\omega^{(out)} = \omega^{(\beta(c))} + \omega^{(\gamma(P))}$. Intuitively, $\omega^{(out)}$ can be treated as a “hot” map for computing $f(x)$ which will affect the decision policy of computing And-node $A$. The higher the score $\omega^{(out)}$ is (i.e., more information propagated from $pg^{(1)}$), the “easier and safer” the acceptance decision could be made. In the same way as Eqn. 5.1, $f(x|\omega^{(out)})$ is factorized,

$$f(x|\omega^{(out)}) = g(x, n|\omega^{(out)}) + h(x, \neg n|\omega^{(out)}) \quad (6.3)$$
6.4.2 The Context-Sensitive Decision Policy Learnt from Redistributed Data by Parse Graphs

In Chapter 5, the near-optimal decision policy of computing an And-node $A$ by its $\alpha$ scoring function $f(x)$ is learnt by solving a constrained optimization problem (Eqn. 5.9) using the $\alpha$ training data $D_\alpha = D_\alpha^+ \cup D_\alpha^-$. From the derivation of the risk function (Eqn. 5.16 and Eqn. 5.17), the population of the distributed data (i.e., the single, pairwise and cumulative pairwise terms defined in Sec. 5.2.3 in Chapter 5) actually define the decision policy. To show that, Eqn. 5.17 is reproduced here,

$$\text{Risk}(\Pi_N, C_{FP}, C_{FN}) = E[n; \Pi_N] + p(\text{FP}; \Pi_N) \cdot \frac{|D^-|}{|D|} \cdot C_{FP} + p(\text{FN}; \Pi_N) \cdot \frac{|D^+|}{|D|} \cdot C_{FN}$$

where $D^- = D_\alpha^-$ and $D^+ = D_\alpha^+$ in learning the decision policy of the $\alpha$ process. The expected computing cost $E[\text{Cost}(\Pi_N)]$ (Eqn. 5.10), the FR probability $p(\text{FP}; \Pi_N)$ (Eqn. 5.12) and the FN probability $p(\text{FN}; \Pi_N)$ (Eqn. 5.14) are defined based on the population of data distributed by a decision policy. Furthermore, $D^-_\alpha$ is often much larger than $D^+_\alpha$ in object detection, and then play a dominant role in minimizing the risk.

By incorporating $\omega^{(\text{out})}$ in computing $f(x)$, it is to consider the $\alpha$-$\beta$-$\gamma$ training data jointly (see Eqn. 3.2, Eqn. 3.3 and Eqn. 3.4) to learn the decision policy with the same formulation used. The difference is that when conditioned on $\omega^{(\text{out})}$, $D^-$ is changed largely but $D^+$ is much more stable. Put in another way, $D^-$ given $\omega^{(\text{out})}$ is the subset of $D^-_\alpha$ where corresponding outside domain represents the “hard negatives” for $pg^{(1)}$. Then, the population size of $D^+$ and $D^-$ become more compatible so that all the single, pairwise and cumulative pairwise terms are redistributed accordingly. Furthermore, after more and more nodes are computed

99
in \( pg^{(1)} \), \( D^- \) shrinks to the joint boundary area defined by computing processes of nodes in \( pg^{(1)} \), and \( D^+ \) may become larger than \( D^- \). So, the decision policy is also adapted with respect to the redistributed data. When \( pg^{(1)} \) is empty, the context-sensitive decision policy recover the context-free one in Chapter 5.

In practice, \( \omega^{(out)} \) is discretized into \( L + 1 \) levels representing the “hotness” (i.e., ignoring the specific structure of \( pg^{(1)} \) for simplicity). Denote by bold \( \Pi_N(\kappa) \) the context-sensitive decision policy of \( f(x|\omega^{(out)} \in Bin(\kappa)) \). Denote by \( D^+_\kappa \) and \( D^-_\kappa \) the positive and negative training data with \( \omega^{(out)} \) being at the \( \kappa \)-th level, and by \( D_\kappa = D^+_\kappa \cup D^-_\kappa \) the whole training data (\( \kappa \in [0, L] \), when \( \kappa = 0 \), \( \Pi_N(\kappa) \) is the same as the context-free decision policy \( \Pi_N \)). The learning of \( \Pi_N(\kappa) \) given \( D_\kappa \) follows the same formulation and procedure presented in Chapter 5.

**Context-Sensitive Training Data for the \( \alpha \) Computing Process.** To generate the dataset \( D^+_\kappa \) and \( D^-_\kappa \), the \( \alpha-\beta-\gamma \) training data are considered jointly and \( pg^{(1)} \) enumerates all combinations of the \( \beta-\gamma \) processes to account for situations occurred during on-line computing. Concretely, given a set of positive images with annotated parse graphs \( \{(I_1, pg_1), \cdots, (I_m, pg_m)\} \) (see Eqn. 3.1), \( \omega^{(out)}_i \)'s are computed for each \( I_i \) with different \( pg^{(1)} \) used (e.g., starting from the cases that \( pg^{(1)} \) includes only one other node than node \( A \), to the cases that \( pg^{(1)} \) consists of all other nodes except node \( A \)). All the computed \( \omega^{(out)}_i \)'s are then discretized into \( L \) levels. For each level \( \kappa \), the positive training data \( D^+_\kappa \) is then collected. To collect \( D^-_\kappa \), given a set of negative images (i.e., images where no instances of the AoG appear) \( \{J_1, \cdots, J_j\} \), the integration of computing processes of nodes in \( pg^{(1)} \) are run using similar procedure as presented in Alg. 1 in Chapter 4 to mine “hard negatives” for \( pg^{(1)} \) at the \( \kappa \)-th level, which compose \( D^-_\kappa \).
6.5 Heuristics for Exploring Parse Graphs

As stated in Sec. 6.3 in Chapter 5, the learnt near-optimal context-sensitive decision policies $\Pi_N(\kappa)^*$ can be used as heuristics to explore a parse graph (i.e., which node should be tested next, the second question proposed in Sec. 6.2 in this chapter).

For notational simplicity, denote by $f(pg)$ the scoring function of a parse graph $pg$ (see Eqn. 4.12), which can also be factorized into two parts: the to-arrive score for nodes already computed, denoted by $g(pg^{(1)})$, and the to-go scores for nodes not tested yet, denoted by $h(pg^{(0)})$,

$$f(pg) = g(pg^{(1)}) + h(pg^{(0)})$$ (6.4)

During on-line computing, for a set of partial parse graph proposals $pg$’s, the scheduling algorithm needs to address the computing order problem: (i) Which parse graph proposal should be selected to explore further and (ii) Which node in the remaining part of the selected parse graph (i.e., $pg^{(0)}$) should be tested? For each parse graph candidate $pg$ and each And-node $A \in V_{And}$, the corresponding context-sensitive decision policy $\Pi_N(\kappa)$ with $g(pg^{(1)}) \in Bin(\kappa)$ is used to compute the heuristics as stated in Sec. 6.3 in Chapter 5 before node $A$ are actually tested. These heuristics will be used to prioritize the agenda and chart in the agenda-chart scheduling algorithm to be presented next in this chapter.

6.6 The Scheduling Space

Given an AoG $\mathcal{G}$ and a testing image $I_\Lambda$, as illustrated in Fig.6.1, the scheduling space $\Psi$ is defined as the product space of the computing process space $\Gamma$ from
The pose space

The computing process space

Figure 6.1: Illustration of the scheduling space consisting of the space of BU/TD computing processes (the right figure) and the pose space (the left figure, i.e., the space of locations in a testing image pyramid). Each terminal node represents the appearance model \( \theta_{\text{app}} \) which is used to evaluate the score in terms of the appearance feature \( F_{\text{app}} \) when placing it in the pyramid.

the AoG and the pose space \( \Phi \) from the image \( I_\Lambda \),

\[
\Psi = \Gamma \times \Phi
\]

where \( \Gamma \) consists of all the \( \alpha\)-\( \beta\)-\( \gamma \) processes in the AoG,

\[
\Gamma = \{(A, \text{Tst}(A)); \forall A \in V_N \cup V_T, \text{Tst} = \alpha | \beta | \gamma\}
\]

and \( \Phi \) is the pose space (Eqn. 6.1).

To specify the deformation model in the \( \beta \) process, i.e., \( p(X(C_i)|X(A)) \) in Eqn. 4.14, let \( \Delta \) be the displacement space for each \( \phi \in \Phi \), and define the displacement operation in the pose space by \( \oplus: \Phi \oplus \Delta \rightarrow \Phi \). For the normal distribution used for modeling \( p(X(C_i)|X(A)) \) (see Eqn. 3.28), it is equivalent to
define a quadratic function of the displacement \( \delta = (dx, dy) \), i.e., the deformation model is defined by \( F_{\text{def}}(\delta) = [dx^2, dx, dy^2, dy]' \).

The basic scheduling element is then defined by,

\[
\psi = (A, \text{Tst}(A), \phi) \quad \text{where} \quad (A, \text{Tst}(A)) \in \Gamma, \phi \in \Phi \quad (6.7)
\]

For each \( \psi \in \Psi \), an auxiliary variable is introduced to indicate whether it is tested or not,

\[
\mu_\psi = \begin{cases} 
1, & \text{if } \psi \text{ is evaluated} \\
0, & \text{otherwise}
\end{cases} \quad (6.8)
\]

and for each \( \phi \in \Phi \) an auxiliary variable is used to indicate the completeness of testing the pose element \( \phi \) (i.e., whether all the computing processes are already evaluated at \( \phi \))

\[
\mu_\phi = \prod_{\psi' : \phi' = \phi} \mu_\psi' \quad (6.9)
\]

6.7 The Agenda-Chart Scheduling Algorithm

During on-line computing in object detection with AoG, we partition the scheduling space \( \Psi \) into two parts \( \Psi = \Psi^{(1)} \cup \Psi^{(0)} \) where \( \Psi^{(1)} = \{ \psi : \mu_\psi = 1, \psi \in \Psi \} \) is the to-arrive subspace where the computing processes are evaluated and \( \Psi^{(0)} = \{ \psi : \mu_\psi = 0, \psi \in \Psi \} \) the to-go subspace where elements are not tested yet, and \( \Psi^{(1)} \cap \Psi^{(0)} = \emptyset \). Furthermore, (i) The to-arrive subspace \( \Psi^{(1)} \) consists of two constituents: the inside subset of accepted elements, denoted by \( \Psi^{(1)}_{\text{in}} \), and the outside subset of rejected elements, denoted by \( \Psi^{(1)}_{\text{out}} \). We have \( \Psi^{(1)} = \Psi^{(1)}_{\text{in}} \cup \Psi^{(1)}_{\text{out}} \) and \( \Psi^{(1)}_{\text{in}} \cap \Psi^{(1)}_{\text{out}} = \emptyset \); and (ii) The to-go subspace \( \Psi^{(0)} \) also contains two terms: the inside subset of elements predictable from elements in \( \Psi^{(1)}_{\text{in}} \) in terms of the AoG, denoted by \( \Psi^{(0)}_{\text{in}} \), and the outside subset of elements not predictable yet,
denoted by $\Psi_{out}^{(0)}$. We have $\Psi^{(0)} = \Psi_{in}^{(0)} \cup \Psi_{out}^{(0)}$ and $\Psi_{in}^{(0)} \cap \Psi_{out}^{(0)} = \emptyset$. So, we have the partition of the scheduling space,

$$\Psi = (\Psi_{in}^{(1)} \cup \Psi_{out}^{(1)}) \cup (\Psi_{in}^{(0)} \cup \Psi_{out}^{(0)})$$

(6.10)

In detection, the scheduling space is partitioned sequentially.

By analogous to the agenda-chart parsing algorithm used in the NLP literature [Caraballo and Charniak, 1998], we introduce the chart to handle the to-arrive inside term $\Psi_{in}^{(1)}$, and the agenda to manage the to-arrive outside elements $\Psi_{out}^{(1)}$ and the to-go terms $\Psi^{(0)}$. Then we have,

$$\Psi = \underbrace{\Psi_{in}^{(1)}}_{\text{chart}} \cup \underbrace{(\Psi_{out}^{(1)} \cup \Psi_{in}^{(0)} \cup \Psi_{out}^{(0)})}_{\text{agenda}}$$

(6.11)

So, we define the risk in the scheduling space as,

$$\text{Risk}(\Psi) = \text{Risk}(\text{Chart}) + \text{Risk}(\text{Agenda})$$

(6.12)

where we have,

$$\text{Risk}(\text{Chart}) = \text{Risk}(\Psi_{in}^{(1)})$$

(6.13)

$$= \sum_{\psi \in \Psi_{in}^{(1)}} \text{Cost}(\psi) + \sum_{A \in \Psi_{in}^{(1)}} C_{A}^{FP} \cdot p(FP|\omega(A, \phi))$$

Note that $p(FP|\omega(A, \phi))$ is estimated using the score map $\omega(A, \phi)$ which integrates the scores terms for the $\alpha$-$\beta$-$\gamma$ computing processes.

$$\text{Risk}(\text{Agenda}) = \text{Risk}(\Psi_{out}^{(1)}) + \text{Risk}(\Psi^{(0)})$$

(6.14)

and the risk of to-arrive outside elements is,

$$\text{Risk}(\Psi_{out}^{(1)}) = \sum_{\psi \in \Psi_{out}^{(1)}} \text{Cost}(\psi) + \sum_{A \in \Psi_{out}^{(1)}} C_{A}^{FN} p(FN|\omega(A, \phi))$$
To estimate the risk of to-go subspace, we first top-down predict the current unobserved elements for each accepted instance in $\Psi_{in}^{(1)}$ based on the AoG (similar to the prediction step in the Early parsing algorithm used in NLP). For those predicted components, we have four situations:

(i) The predicted components are outside the pose space, and then they are not considered any more.

(ii) The predicted components are placed in $\Psi_{in}^{(1)}$, and therefore conflicted with some existing instances. We need resolve the conflict.

(iii) The predicted components are placed in $\Psi_{out}^{(1)}$ (where we have the outputs of some tests).

(iv) The predicted components are in $\Psi^{(0)}$ (where is completely unobserved so far).

To resolve the conflicts in the second situation, if had, we first estimate new scores for those augmented instances and the probabilities of being FNs and FPs, and then we accept the predicted components and remove the existing conflicted instances if the overall risk decreased (i.e. the decreased risk in the to-arrive inside risk is larger than the increased risk in the to-arrive outside risk). Otherwise, we reject the predicted components.

If no predicted components could be placed correctly in the pose space at all, we need to either explore $\Psi^{(0)}$ if there are still some $\mu_\phi = 0$ (i.e., testing not complete yet), or the scheduling stops. Otherwise, we obtain a set of updated instances, denoted by $A_{i,*}^t$ (i.e. instances augmented with predicted components), and by $pg^*$ the completion version of the parse graph $pg$. Then, we have,

$$\text{Risk}(\Psi^{(0)}) = \text{Risk}(\Psi_{in}^{(0)}) + \text{Risk}(\Psi_{out}^{(0)}) \quad (6.16)$$
and Risk($\Psi^{(0)}_m$) and Risk($\Psi^{(0)}_o$) can be estimated in terms of the decision policies learnt in Chapter 5. So, the risk of agenda can be estimated.

6.8 Summary

The agenda-chart scheduling algorithm presented in this chapter is a theoretical study under the best-first heuristic search framework. The experiments will be done in the future work.
CHAPTER 7

Conclusions and Discussions

7.1 Thesis Summary

This thesis presents a framework of integration and goal-guided scheduling of BU/TD computing processes in a recursively defined hierarchical AoG to address the vision-goal-guided trade-off between accuracy performance and computational efficiency. The study consists of five parts: (i) Identifying three types of BU/TD computing processes for each And-node $A$ in an AoG, termed the $\alpha$-$\beta$-$\gamma$ computing processes, and learning the three processes under MLE separately by using a proposed isolation method; (ii) Evaluating the individual information contributions of BU/TD processes in terms of the discriminative power with both computer and human experiments; (iii) Integrating BU/TD computing processes by BFS for robust inference in object parsing using AoG formulated under the Bayesian framework; (iv) Learning near-optimal decision policies taking account for both computing costs and FP and FN loss costs to speed up the computing processes of terminal nodes and And-nodes in an AoG given allowable bounds on accuracy performance in cost-sensitive object detection/parsing; and (v) Scheduling BU/TD computing processes by best-first heuristic search using the learnt decision policies and with an agenda-chart scheduling algorithm.
7.2 Representational and Computational Knowledge

In the literature of computer vision, accuracy performance and computing efficiency are two of the most important issues. Accuracy performance is addressed by exploring different kinds of representational knowledge which mainly includes how to build up a visual vocabulary, model object structures and relations and estimate parameters. Much work has been done in which hierarchical models and contextual information have shed lights on advancing the accuracy performance [Grenander and Miller, 2007; Geman et al., 2002; Riesenhuber and Poggio, 1999; Ullman et al., 2002; Todorovic and Ahuja, 2008b; Wu et al., 2010; Sudderth et al., 2008; Felzenszwalb et al., 2010; Fidler et al., 2008; Torralba, 2003; Divvala et al., 2009]. On the other hand, computing efficiency is addressed by utilizing different types of computational knowledge which often consists of two types, one is “algorithmic” or “ensemble-independent” and the other “ensemble-targeting”. The “algorithmic” method exploits some mathematical optimization techniques for the objective function learned from an ensemble and is generic. In recent literature of computer vision, a few efforts are made for the “algorithmic” method such as the branch-and-bound method [Lampert et al., 2009] which assumes general lower and upper bounds can be derived for the objective function. In contrast, the “ensemble-targeting” method aims to learn to compute faster by taking the computing order into account, i.e., active computing. Very few work has been devoted to active computing [Blanchard and Geman, 2005]. The scheduling problem addressed in this thesis is “ensemble-targeting” based on the AoG representation.
7.3 Future Work

There are a few directions which worth exploring further based on the current research in this thesis in the future work.

(1) Evaluating the BU/TD computing processes trained with different types of features. BU/TD computing processes are defined by both appearance evaluation schema and the geometric compatibility function (e.g., the $\beta$ binding model). The appearance features used in this thesis are mainly based on active basis model with Gabor filter responses, as well as the Haar features in the Adaboost classifier. The geometric model is assumed to be Gaussian distribution which is often too simple to capture more meaningful constraints in articulated objects.

(2) Extending the integration of BU/TD computing processes to deep hierarchical model with And-Or structures. When the hierarchical model has deep structures, the learning of the BU/TD computing processes will be an important issue, especially how to estimate the parameters jointly to make the integration of BU/TD computing processes in a principled way.

(3) Learning decision policies for Or-nodes with different model structures and even the whole AoG. The decision policy learnt in this thesis is based on the sequential factorization of the scoring function of a computing process. For Or-nodes and even a portion of AoG, the scoring function is often non-linear and then the factorization is not straight forward.

(4) Studying the scheduling algorithm in real benchmark dataset of object detection tasks.


