UNIVERSITY OF CALIFORNIA

Los Angeles

Sampling and Learning of the And-Or Graph

A thesis submitted in partial satisfaction

of the requirements for the degree

Master of Science in Statistics

by

Ruize Zhang

2020

ABSTRACT OF THE THESIS

Sampling and Learning
of the And-Or Graph

by

Ruize Zhang
Master of Science in Statistics
University of California, Los Angeles, 2020
Professor Yingnian Wu, Chair

The And-Or graph is a tool for knowledge representation. In this thesis we first study the sampling of the And-Or graph with or without context constraints. Without any constraint on the potential functions of the And-Or graph nodes, the positions and shapes of different components of the face images are not aligned properly. In contrast, with both unary constraints and binary constraints, the components are aligned and the samples are more representative of the And-Or graph. We further explore parameter and structure learning of the And-Or graph by implementing and applying some existing algorithms. The experimental results on 1D text data and 2D face image data are shown. While there is no apparent difference between the sampling results of the parameter learned And-Or graph and the true And-Or graph, the sampling results of the structure learned And-Or graph are not perfect and could be further improved.

To my family

TABLE OF CONTENTS

1 Introduction **1**

1.1 Grammar 1

1.2 Parse Graph 2

1.3 And-Or Graph 3

1.4 Content Overview 4

2 Sampling of the And-Or Graph **5**

2.1 Unconstrained Sampling 5

2.2 Constrained Sampling with Unary Potential Function 8

2.3 Constrained Sampling with Binary Potential Function 11

3 Parameter Learning of the And-Or Graph **13**

4 Structure Learning of the And-Or Graph **17**

References **23**

LIST OF FIGURES

1.1 Example of a parse tree of a sentence 2
1.2 Example of a AOG tree structure 3

2.1 AOG 1D text 6
2.2 AOG 2D face 6
2.3 Unconstrained sample of 1D text 7
2.4 Unconstrained sample of 2D face 7
2.5 Unary-constrained sample of 1D text 10
2.6 Unary-constrained sample of 2D face 10
2.7 Binary-constrained sample of 1D text 12
2.8 Binary-constrained sample of 2D face 12

3.1 Parameter learned sample of 2D face 15
3.2 True face grammar in parameter learning 16
3.3 Learned face grammar in parameter learning 16

4.1 Structure learning training data: 1D text 20
4.2 Structure learning training data: robotic commands 20
4.3 The learning algorithm in [TPZ13] 21
4.4 Sample of 1D text generated by learned grammar 22
4.5 Sample of robotic commands generated by learned grammar 22

ACKNOWLEDGMENTS

The code for the algorithms mentioned in this thesis is based on the template code of the project assignments in the course Statistics M232B: Statistical Computing and Inference in Vision and Cognition.

CHAPTER 1

Introduction

1.1 Grammar

In real world signals like natural languages or images, a grammar can be used to describe how the basic elements can be grouped together to form larger parts of the signals. We can observe that many small parts appear together very often. For example, a noun-phrase is a "reusable" part in language. Such parts repeated a lot as building blocks of more complex sentences. According to [Cho57], a grammar can be formulated as $G = (V_N, V_T, R, S)$. In this formulation, V_N represents the parts of the signals that can be further separated into smaller parts. V_T represents the most elementary parts, R means the rules of generating those smaller parts from higher lever parts and S stands for the root for us to expand. As an illustration of that, we can define a simple grammar ourselves:

$S \to M$

$M \to N$

$M \to N + / - N$

$N \to 0/1/2/3/3/5/6/7/8/9$

In the example above, S is the starting point, the rules are represented by arrows, and M and N are non-terminal nodes. The plus and minus tokens and numbers from 0 to 9 are terminal nodes.

1.2 Parse Graph

Parsing can be regarded as the way of analyzing or interpreting a signal and a parse tree can be used to represent how the signals are separated recursively into reusable parts and then more elementary parts. Each non-terminal node of the parse tree can be replaced by its children until the leaf nodes are reached. A parse graph can be regarded as the combination of a parse tree to represent the grammar structure introduced in Section 1.1 and also the relations between the vertices, which are defined as $\{(v, u)\} \subset V \times V$. Here V denotes the vertex set. For example, the relations can be the chance of co-occurrence of words in text and the orientations and relative positions for different components of images. An example parse tree of a sentence is given in Figure 1.1.

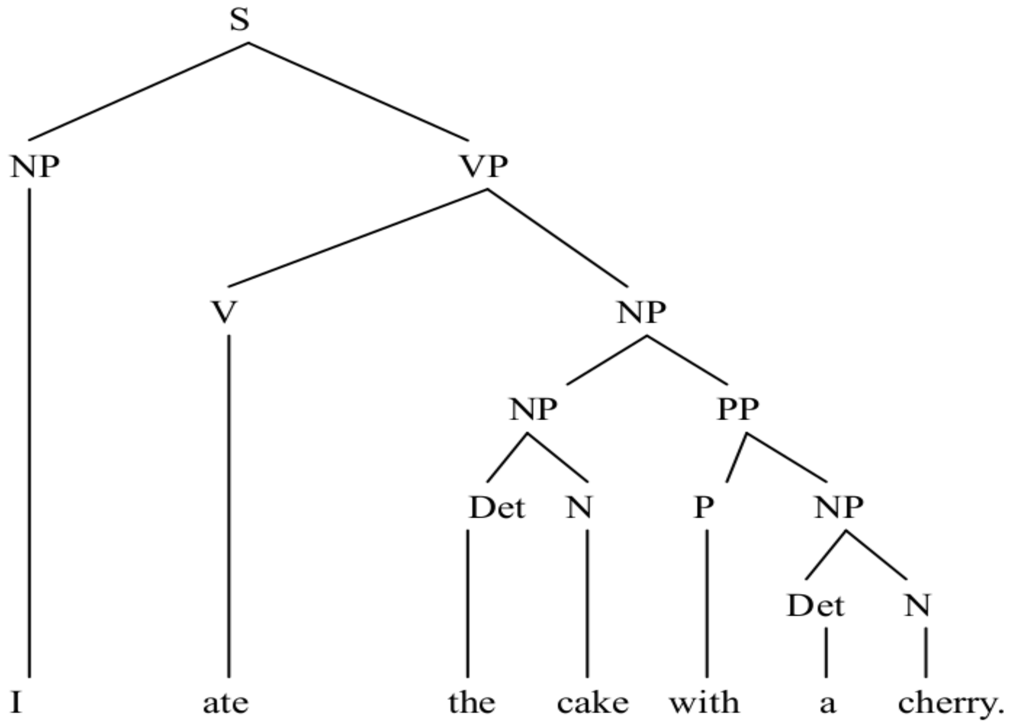


Figure 1.1: Example of a parse tree of a sentence

1.3 And-Or Graph

In parse graphs, there is no or-node to represent substitutions between multiple components. To represent this relation of substitution in the possible parse graphs, we can use the And-Or graph. An And-Or graph contains and-nodes to combine different parts, or-nodes for parts that can be substituted and leaf nodes at the bottom as basic elements of the signals. Each of the children of the or-nodes has a weight associate with it, determining the probability that this specific child being chosen. An And-Or Graph can also have links between nodes of the same level to describe the relation between different parts of the same level of the graph. An example And-Or tree structure is given in Figure 1.2. Given that an And-Or graph is a recursive structure that can potentially contain multiple or-nodes in each level, it has strong expressive power to generate large amount of combinations of the leaf nodes using relatively small number of non-terminal and-nodes and or-nodes. As an efficient way of knowledge representation, it is first explicitly used in [CXL06]. In Statistics M232B: Statistical Computing and Inference in Vision and Cognition, spatial, temporal, causal and attribute And-Or graphs are introduced for different objects of modeling and in this thesis we only consider the application of the And-Or graph in modeling text and images.

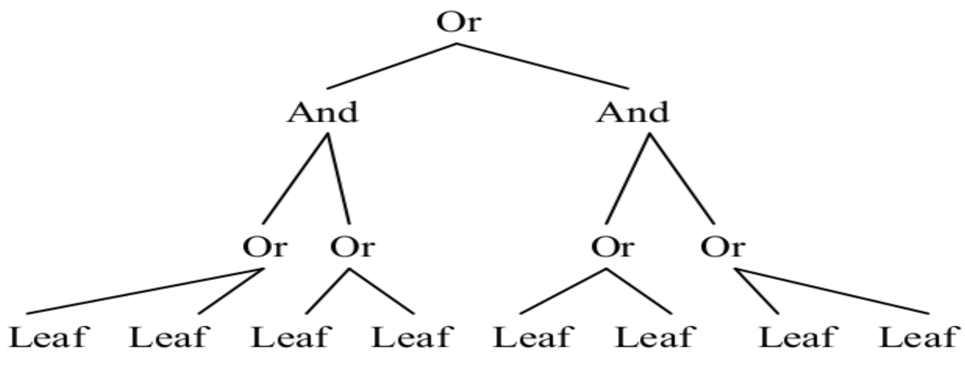


Figure 1.2: Example of a AOG tree structure

1.4 Content Overview

In the following chapters, we first explore the sampling of the And-Or graph with or without context constraints. As we can see from the sampling results of the face images, if we don't consider the constraints on the potential functions of the And-Or graph nodes, the positions and shapes of different components of the face image are not aligned properly. In contrast, when we add both unary constraints and binary constraints, the components are aligned and we the samples are more representative of the And-Or graph. Then in chapter 3 and 4 we explore parameter and structure learning of the And-Or graph by implementing and applying the algorithms introduced in [HNF18] and [TPZ13]. The experimental results on 1D text data and 2D face image data are shown. While there is no apparent difference between the sampling results of the parameter learned And-Or graph and the true And-Or graph, the sampling result of the structure learned And-Or graph is not perfect and may be further improved as suggested in the conclusion of chapter 4.

CHAPTER 2

Sampling of the And-Or Graph

2.1 Unconstrained Sampling

An And-Or graph can generate multiple configurations, which are the sequences formed by all of its leaf nodes. In this section, we sample from all the possible configurations of the And-Or graph. Since the weights of the or-nodes represent the likelihood for their children to appear, if we randomly choose one child of each or-node according to its weight and choose all children of each and-node, we can get the unconstrained sampling result, which turns out to be a single parse tree. Here we use two examples to illustrate the sampling, one is the sampling from an And-Or Graph for 1d text and the other is for 2d face images. Figure 2.1 shows the And-Or structure of the 1d text. Similarly, Figure 2.2 shows the And-Or structure of the 2d face images. The 1D text And-Or graph defines how the sentences can be composed by the leaf words. And the 2D face images shows how the face image is composed by left eye, right eye, left ear,right ear, nose and mouth of different types. During the sampling process, we start from the top level root node and expand each node in a top-down order. To generate the samples with each component in the proper order, we also need to record the order of occurrence of the ancestors of the leaf nodes and the leaf nodes themselves from left to right. Examples of the sampling result of the text and face are shown in Figure 2.3 and Figure 2.4. Figure 2.3 contains the rules used in generating the sequence of text and then the text itself. Given that this is unconstrained sampling, we do not consider the attributes and relations associate with the nodes. So we can see that for each rule in Figure 2.3, the two attributes of each node are randomly selected. And in Figure 2.4 the parts of the face are not aligned properly.

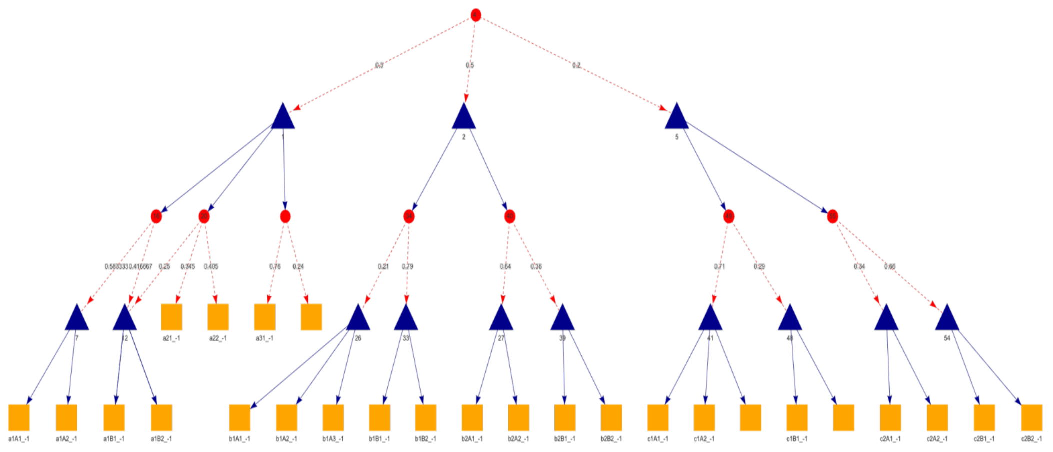


Figure 2.1: AOG 1D text

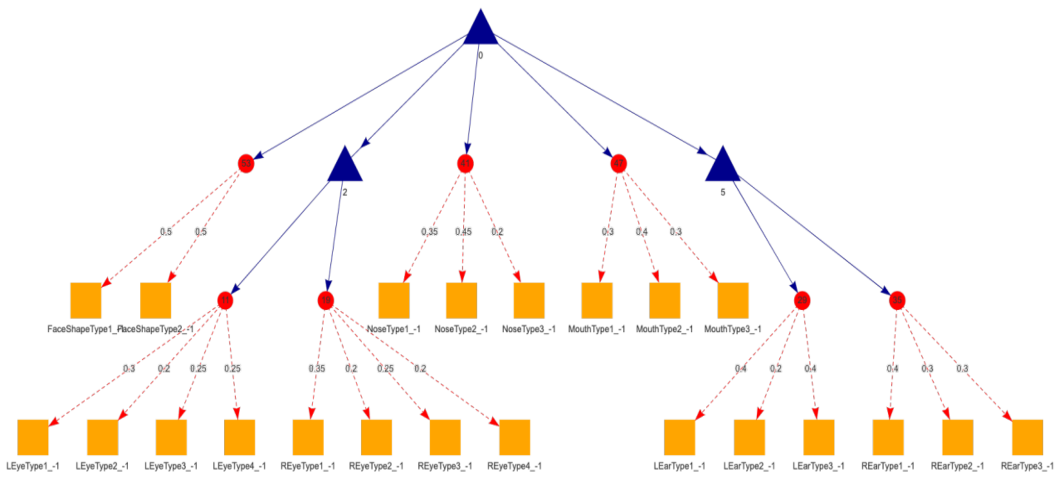
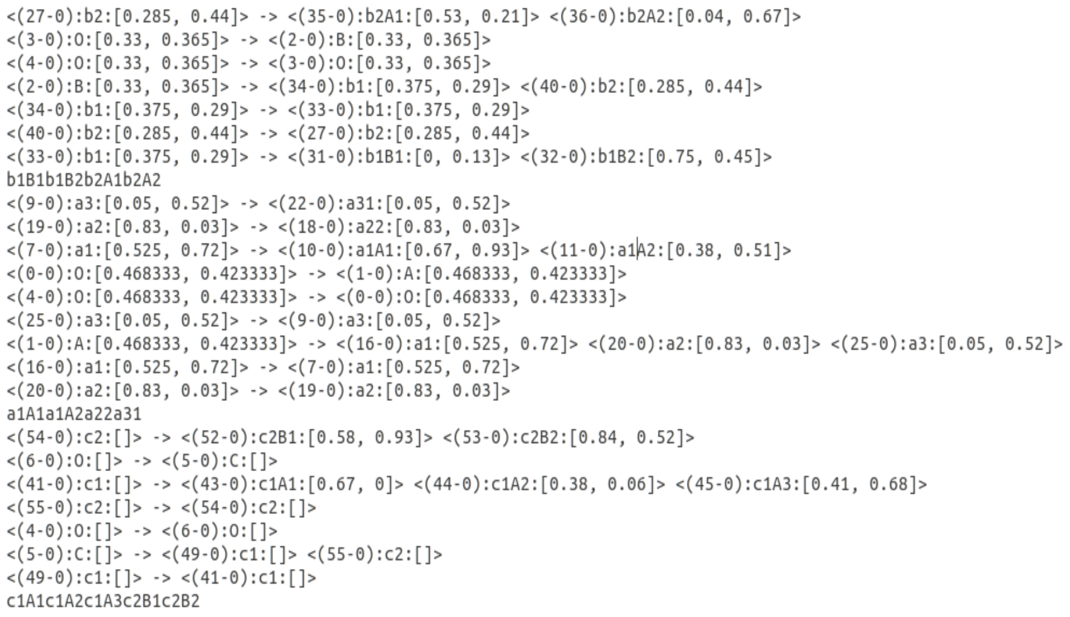


Figure 2.2: AOG 2D face



```
<(27-0):b2:[0.285, 0.44]> -> <(35-0):b2A1:[0.53, 0.21]> <(36-0):b2A2:[0.04, 0.67]>
<(3-0):O:[0.33, 0.365]> -> <(2-0):B:[0.33, 0.365]>
<(4-0):O:[0.33, 0.365]> -> <(3-0):O:[0.33, 0.365]>
<(2-0):B:[0.33, 0.365]> -> <(34-0):b1:[0.375, 0.29]> <(40-0):b2:[0.285, 0.44]>
<(34-0):b1:[0.375, 0.29]> -> <(33-0):b1:[0.375, 0.29]>
<(40-0):b2:[0.285, 0.44]> -> <(27-0):b2:[0.285, 0.44]>
<(33-0):b1:[0.375, 0.29]> -> <(31-0):b1B1:[0, 0.13]> <(32-0):b1B2:[0.75, 0.45]>
b1B1b1B2b2A1b2A2
<(9-0):a3:[0.05, 0.52]> -> <(22-0):a31:[0.05, 0.52]>
<(19-0):a2:[0.83, 0.03]> -> <(18-0):a22:[0.83, 0.03]>
<(7-0):a1:[0.525, 0.72]> -> <(10-0):a1A1:[0.67, 0.93]> <(11-0):a1A2:[0.38, 0.51]>
<(0-0):O:[0.468333, 0.423333]> -> <(1-0):A:[0.468333, 0.423333]>
<(4-0):O:[0.468333, 0.423333]> -> <(0-0):O:[0.468333, 0.423333]>
<(25-0):a3:[0.05, 0.52]> -> <(9-0):a3:[0.05, 0.52]>
<(1-0):A:[0.468333, 0.423333]> -> <(16-0):a1:[0.525, 0.72]> <(20-0):a2:[0.83, 0.03]> <(25-0):a3:[0.05, 0.52]>
<(16-0):a1:[0.525, 0.72]> -> <(7-0):a1:[0.525, 0.72]>
<(20-0):a2:[0.83, 0.03]> -> <(19-0):a2:[0.83, 0.03]>
a1A1a1A2a22a31
<(54-0):c2:[]> -> <(52-0):c2B1:[0.58, 0.93]> <(53-0):c2B2:[0.84, 0.52]>
<(6-0):O:[]> -> <(5-0):C:[]>
<(41-0):c1:[]> -> <(43-0):c1A1:[0.67, 0]> <(44-0):c1A2:[0.38, 0.06]> <(45-0):c1A3:[0.41, 0.68]>
<(55-0):c2:[]> -> <(54-0):c2:[]>
<(4-0):O:[]> -> <(6-0):O:[]>
<(5-0):C:[]> -> <(49-0):c1:[]> <(55-0):c2:[]>
<(49-0):c1:[]> -> <(41-0):c1:[]>
c1A1c1A2c1A3c2B1c2B2
```

Figure 2.3: Unconstrained sample of 1D text

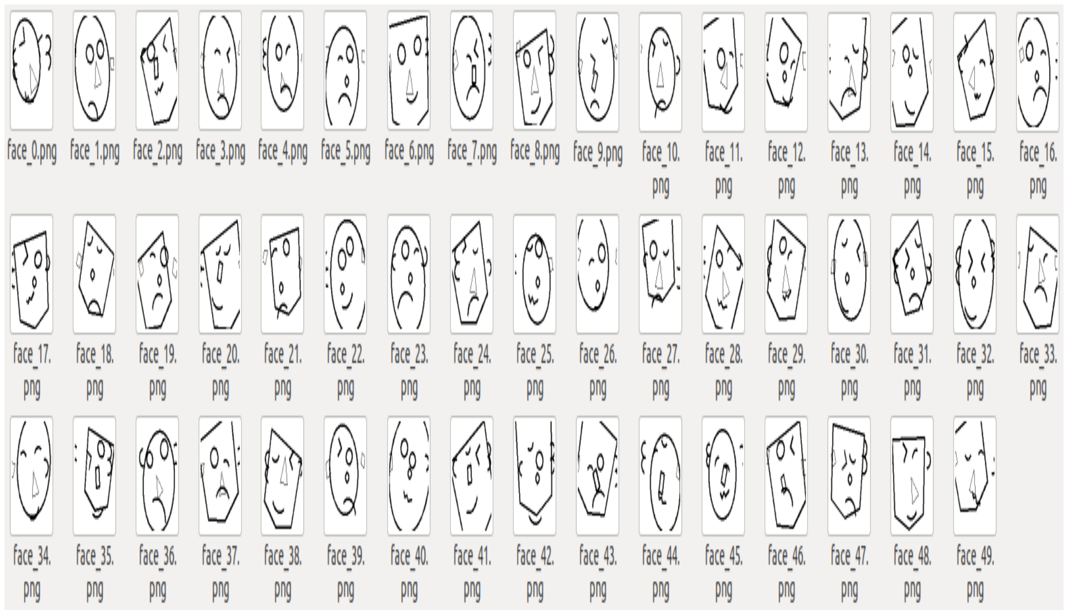


Figure 2.4: Unconstrained sample of 2D face

2.2 Constrained Sampling with Unary Potential Function

Without the constraints for the attributes of the vertices in the And-Or Graph, we can generate texts or images with correct components. However, as we can see from the unconstrained face images, the components are not arranged in the proper way. So we can introduce complexity into the model using the stochasticity brought by potential functions.

According to the course materials and slides of STATS-232B taught by professor Song-Chun Zhu, using the maximum entropy principle to match the frequency of the or-nodes and other statistics one can get the following probability model on the parse graph contained in the And-Or Graph:

$\varepsilon(pg) = \sum_{v \in V^{or}(pg)} \lambda_v(w(v)) + \sum_{t \in leaf(pg) \cup V^{and}(pg)} \lambda_t(\alpha(t))$

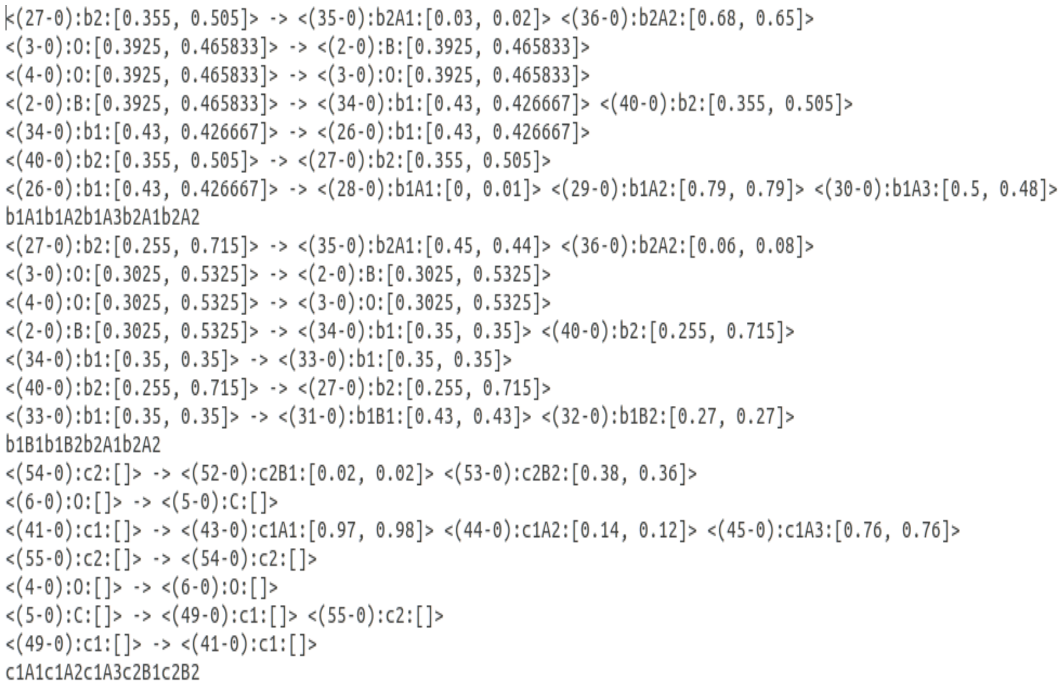
$+ \sum_{(i,j) \in E(pg)} \lambda_{i,j}(v_i, v_j, \gamma_{ij}, \rho_{ij})$

$p(pg) = \frac{1}{Z(\Theta)} exp(-\varepsilon(pg))$

In the equations above, $\varepsilon(pg)$ is the total energy. The first term λ_v in the energy function accounts for the or-node weights, which corresponds to the probability of occurrence for each of its' children. The second and third term are related to the potential function as the constraints in the model. Specifically, the second term λ_t corresponds to energy of a single node, which is related to the attributes $\alpha(t)$ of the node. The third term λ_{ij}, on the other hand, corresponds to the relation between pairs of nodes. γ_{ij} and ρ_{ij} here can be regarded as attributes relating to the structure that binds the pair of vertices and the compatibility of the pair of vertices. In Figure 2.5 and 2.6, the sampling of the same And-Or grammar, from section 2.1, for both the text and images, are shown.

As we can see from the sample of the 1D text, the two attributes of the leaf nodes are now similar to each other. For the 2D face images, the parts of the face are now aligned in better positions but different parts, for example the two eyes or ears, may not be the same type. Also the orientation of the parts may not align with each other.

In the constrained sampling, we first use the unconstrained sampling to generate the parse tree of the And-Or Graph as in chapter 2.1, then use Gibbs Sampling method to modify the attributes of the nodes. To perform Gibbs Sampling, we first randomly initialize the attributes. Since each of the attribute values has different range and is continuous, we can't directly sampling from all possible values from the range. One way we can use is to divide the range of the attribute values into 100 bins. In each modification, we choose one value from the bins. Then in each iteration, we calculate the conditional probability of one single attribute of one leaf node given the attributes of all other attributes, which is proportional to their joint probability. The joint probability is then given by the probability model above. Since for each leaf node there are multiple attributes and for each attribute there are 100 bins, for each leaf node, we need to calculate $\mid attribute \mid *100$ values. After 200 Gibbs iterations, we obtain the results in Figure 2.5 and 2.6.



```
<(27-0):b2:[0.355, 0.505]> -> <(35-0):b2A1:[0.03, 0.02]> <(36-0):b2A2:[0.68, 0.65]>
<(3-0):O:[0.3925, 0.465833]> -> <(2-0):B:[0.3925, 0.465833]>
<(4-0):O:[0.3925, 0.465833]> -> <(3-0):O:[0.3925, 0.465833]>
<(2-0):B:[0.3925, 0.465833]> -> <(34-0):b1:[0.43, 0.426667]> <(40-0):b2:[0.355, 0.505]>
<(34-0):b1:[0.43, 0.426667]> -> <(26-0):b1:[0.43, 0.426667]>
<(40-0):b2:[0.355, 0.505]> -> <(27-0):b2:[0.355, 0.505]>
<(26-0):b1:[0.43, 0.426667]> -> <(28-0):b1A1:[0, 0.01]> <(29-0):b1A2:[0.79, 0.79]> <(30-0):b1A3:[0.5, 0.48]>
b1A1b1A2b1A3b2A1b2A2
<(27-0):b2:[0.255, 0.715]> -> <(35-0):b2A1:[0.45, 0.44]> <(36-0):b2A2:[0.06, 0.08]>
<(3-0):O:[0.3025, 0.5325]> -> <(2-0):B:[0.3025, 0.5325]>
<(4-0):O:[0.3025, 0.5325]> -> <(3-0):O:[0.3025, 0.5325]>
<(2-0):B:[0.3025, 0.5325]> -> <(34-0):b1:[0.35, 0.35]> <(40-0):b2:[0.255, 0.715]>
<(34-0):b1:[0.35, 0.35]> -> <(33-0):b1:[0.35, 0.35]>
<(40-0):b2:[0.255, 0.715]> -> <(27-0):b2:[0.255, 0.715]>
<(33-0):b1:[0.35, 0.35]> -> <(31-0):b1B1:[0.43, 0.43]> <(32-0):b1B2:[0.27, 0.27]>
b1B1b1B2b2A1b2A2
<(54-0):c2:[]> -> <(52-0):c2B1:[0.02, 0.02]> <(53-0):c2B2:[0.38, 0.36]>
<(6-0):O:[]> -> <(5-0):C:[]>
<(41-0):c1:[]> -> <(43-0):c1A1:[0.97, 0.98]> <(44-0):c1A2:[0.14, 0.12]> <(45-0):c1A3:[0.76, 0.76]>
<(55-0):c2:[]> -> <(54-0):c2:[]>
<(4-0):O:[]> -> <(6-0):O:[]>
<(5-0):C:[]> -> <(49-0):c1:[]> <(55-0):c2:[]>
<(49-0):c1:[]> -> <(41-0):c1:[]>
c1A1c1A2c1A3c2B1c2B2
```

Figure 2.5: Unary-constrained sample of 1D text

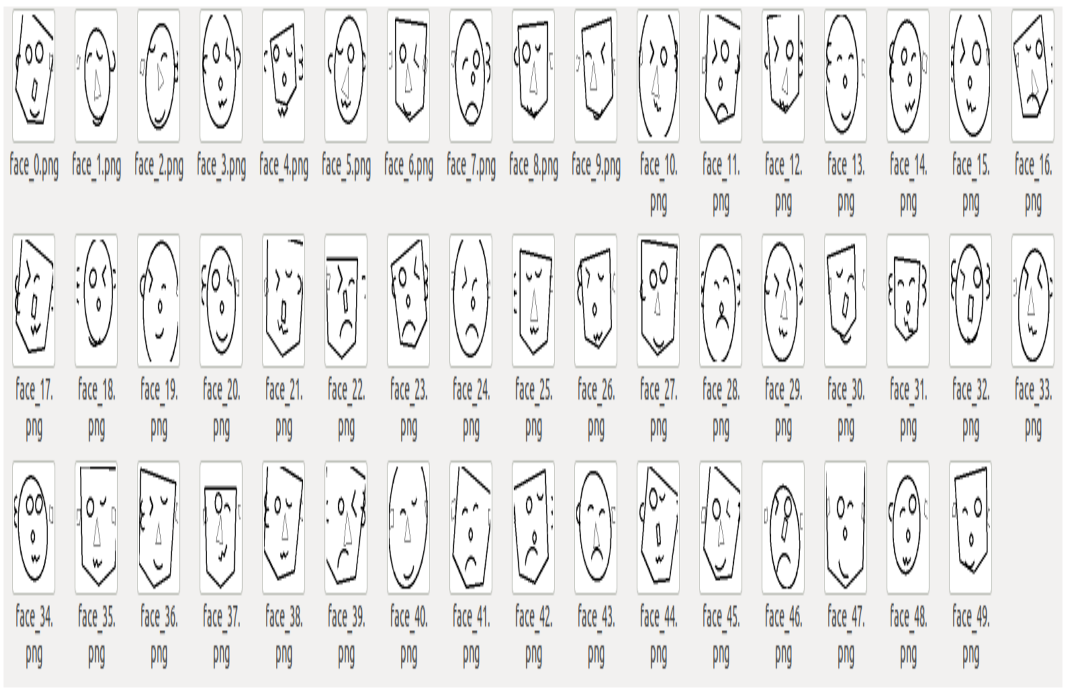
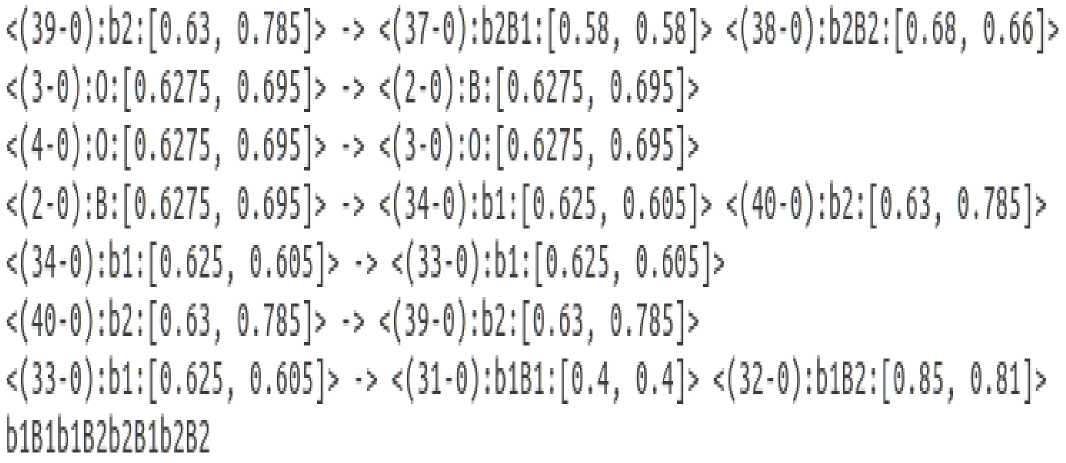


Figure 2.6: Unary-constrained sample of 2D face

2.3 Constrained Sampling with Binary Potential Function

We can further add binary constraints to the sampling. That is, we consider the binary potential functions in the energy equation in Section 2.2. Since the binary potential function is determined by the attributes of pairs of nodes, we need to calculate the attributes of non-terminal nodes which have relations with other non-terminal nodes in addition to the potential of the single leaf node calculated in Section 2.2. In every iteration of the Gibbs Sampling, we again divide the attribute range of each node that has a potential function into 100 bins. Then when calculating the joint probabilities, we add the third term in the probability equation in Section 2.2. Since the attributes of the non-terminal nodes here are set as the sum of their children nodes, we can start from the leaf level and calculate the attributes of non-terminal nodes recursively. Given that different pairs of nodes have different binary potential functions, we set different binary potential parameters according to the semantic states of the nodes and pass that as a parameter to the binary potential function. In the process of the Gibbs Sampling, since we need to calculate the joint probabilities for all possible values of the attributes of the leaf nodes and the joint probability is related to the potential energy of the non-terminal nodes, we need to modify all the attributes of the non-terminal nodes accordingly. Figure 2.7 and 2.8 show the results generated after 200 Gibbs Iterations. This time we can see that the face images generated have all the components in the proper positions and also aligned with each other.

In conclusion, in this chapter we sampled from the And-Or graph for the 1D text and 2D face image. In this process we explored unconstrained, unary constrained and binary constrained sampling. While in the unconstrained sampling we just consider the structure of the And-Or graph, in the constrained sampling we also consider other factors in the energy function, including the unary potential function of the terminal nodes and binary potential functions between pairs of nodes. To sample from the distribution, Gibbs sampling is applied. From the sampling results of the face images, we can see that when both constraints are added, positions and shapes of different components of the faces are aligned properly.



```
<(39-0):b2:[0.63, 0.785]> -> <(37-0):b2B1:[0.58, 0.58]> <(38-0):b2B2:[0.68, 0.66]>
<(3-0):O:[0.6275, 0.695]> -> <(2-0):B:[0.6275, 0.695]>
<(4-0):O:[0.6275, 0.695]> -> <(3-0):O:[0.6275, 0.695]>
<(2-0):B:[0.6275, 0.695]> -> <(34-0):b1:[0.625, 0.605]> <(40-0):b2:[0.63, 0.785]>
<(34-0):b1:[0.625, 0.605]> -> <(33-0):b1:[0.625, 0.605]>
<(40-0):b2:[0.63, 0.785]> -> <(39-0):b2:[0.63, 0.785]>
<(33-0):b1:[0.625, 0.605]> -> <(31-0):b1B1:[0.4, 0.4]> <(32-0):b1B2:[0.85, 0.81]>
b1B1b1B2b2B1b2B2
```

Figure 2.7: Binary-constrained sample of 1D text

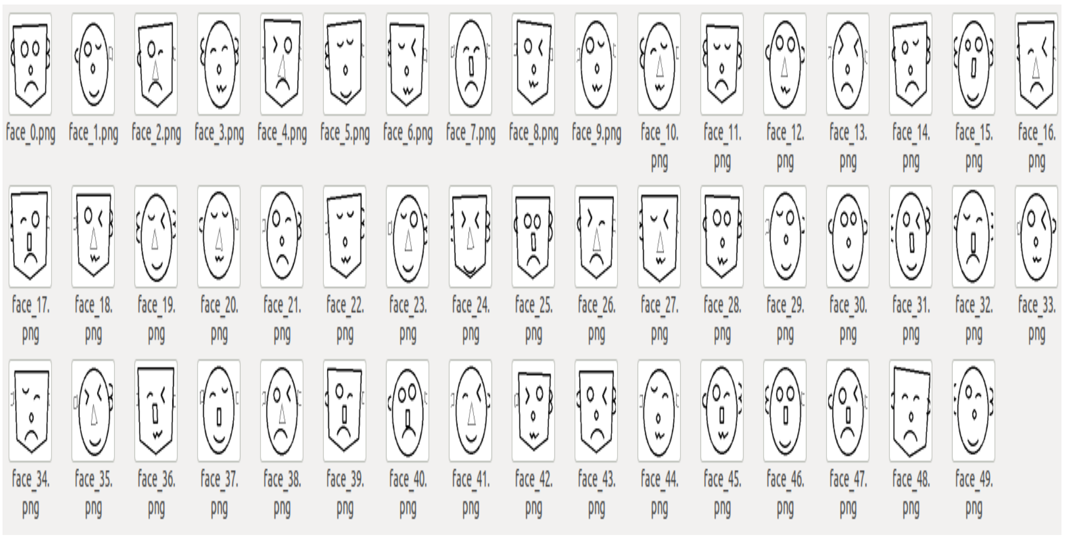


Figure 2.8: Binary-constrained sample of 2D face

CHAPTER 3

Parameter Learning of the And-Or Graph

In the previous sections, the And-Or graph structure and parameters are known. In this section, we continue using the face image example to explore the learning of the And-Or graph parameters with the structure known. The parameters that we want to learn include the or-node weights and potential function parameters. To learn the or-node weights, we just need to count the or-node frequencies in the training data. That is, we count the number of occurrence of the children of each or-node. If we denote the children of one or-node as $V_{or_1}...V_{or_n}$, this formulation is given as: $w(V_{or_i}) = \frac{|V_{or_i}|}{|V_{or}|}$.

In Section 2.2 and 2.3, we have mentioned that the potential is a function of attributes of the nodes. For unary potential function, the potential function of a node is related to the attributes of the node itself. For binary potential function, the potential of a pair of nodes is related to the attributes of this pair. For this face example, we can define 14 variables to represent the mean x and y position of the face, nose, left eye,right eye, left ear, right ear and mouth. These 14 variables are then used in the calculation of the potential function. In this section, we aim to learn the value of those variables. To be more specific, we set those parameters as following: face x mean=64, face y mean=64, nose x mean=64, nose y mean=70, mouth x mean=64, mouth y mean=85, ear left x mean=118, ear left y mean=50, ear right x mean=10, ear right y mean=50, eye left x mean=80, eye left y mean=42, eye right x mean=48, eye right y mean=42. During the learning, we can use the L2 distance of the parameter values we learned and the ground truth values to check the correctness of this procedure. In the experiment, we witness this L2 distance reduces to around 5.

To learn the parameters, we apply the self-critic learning of energy based models in [HNF18]. We use the relation: $\frac{\partial}{\partial\alpha}KL(q_{data}(x) \parallel \pi_\alpha(x)) = E_{q_{data}}[\frac{\partial}{\partial\alpha}f_\alpha(x)] - E_{\pi_\alpha}[\frac{\partial}{\partial\alpha}f_\alpha(x)]$. Here α is the parameter we learn and $f_\alpha(x)$ is the total potential function, i.e., the sum of unary potential and binary potential given in section 2.2. And here we need to minimize the KL divergence between the training sample data distribution and our parameterized probabilistic distribution.

To calculate the averaged potential gradient function of both training and sampled data, we need to first get the sum of the gradient introduced by the potential function of each node or pair of node. For the leaf nodes, the unary potential function is only related to their attributes and the parameters we set. So we can easily compute the gradient using the unary potential function. However, for binary potential function, the gradient is not that easy to compute. To solve this problem, we can apply the numerical way of computing the gradient. Since the parameters in this image case is actually the positions of the components of the face, the value is discrete and the difference between the adjacent values is 1. Therefore, we can calculate the binary potential of parameter and parameter plus 1 and use the difference as the potential gradient. To decide which of the 14 parameters should be used to calculate gradient towards, we can first give each of them a index. A binary potential parameter is passed to each binary potential gradient function, which records the index of the parameters that is used. After we get each single gradient, we add them together as the total potential gradient of one data, which is then averaged over the dataset.

During the experiment, We use 1000 training data and generate samples in each iteration. Specifically in each iteration, we use the current potential function to sample new data and use the average potential function gradient with respect to each of the 14 parameters to compare with that of the training data. Although we have 1000 training data, generating 1000 samples in each iteration can be slow to compute. To reduce the amount of computation we reduce the number of samples to 50. Figure 3.1 shows the samples generated by the learned model. The true grammar and learned grammar are given in Figure 3.2 and Figure 3.3.

In conclusion, in this part we apply the self-critic learning of energy based models in [HNF18] to learn the And-Or graph parameters. From the sampling result, we can see that the positions of the different parts of the face are aligned correctly. And there is no apparent difference between the binary constrained samples we generated in Section 2.3 using the true grammar and the samples we generated using the learned And-Or graph parameters.

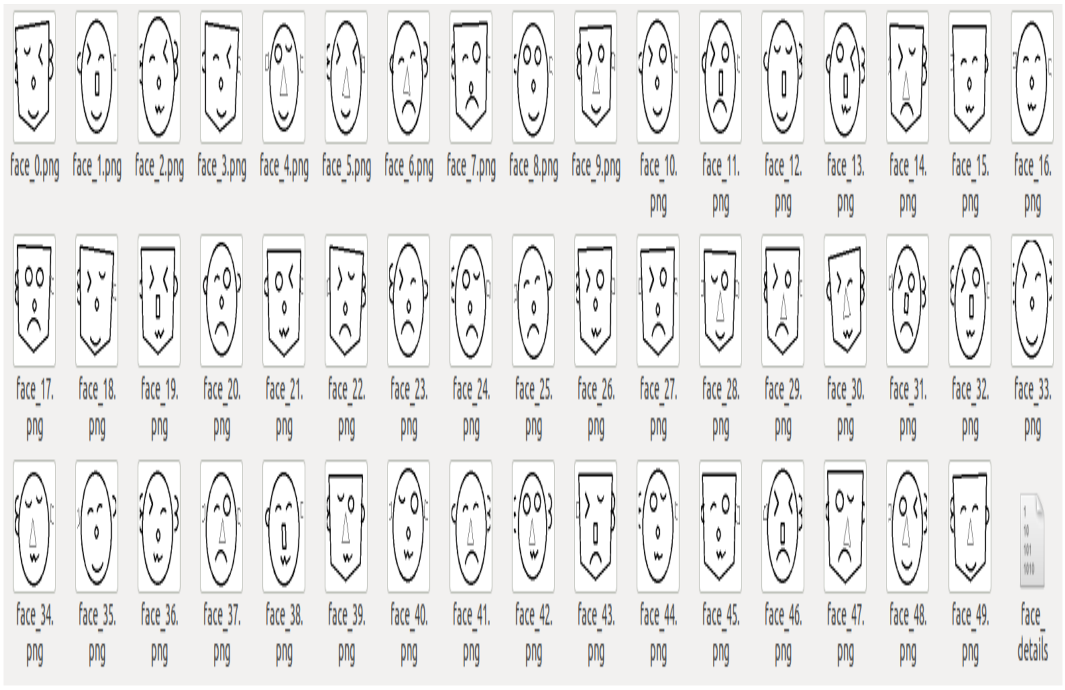
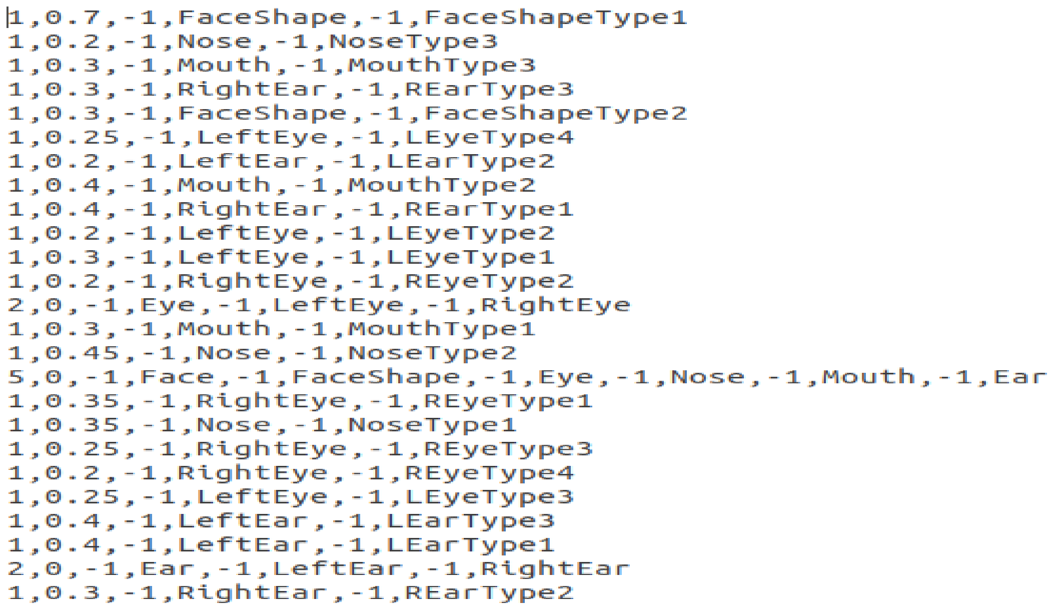
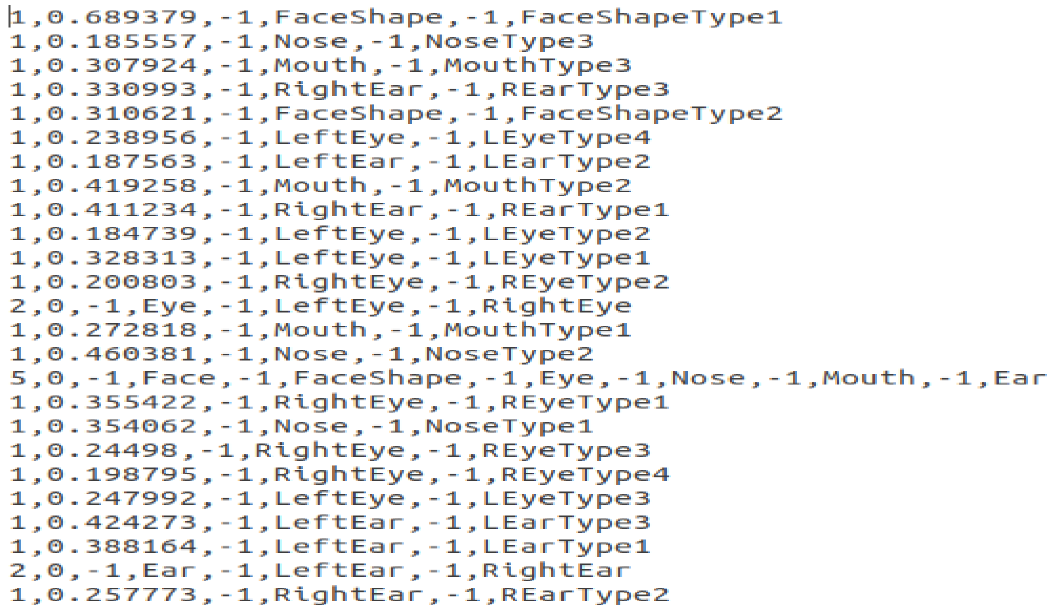


Figure 3.1: Parameter learned sample of 2D face



```
1,0.7,-1,FaceShape,-1,FaceShapeType1
1,0.2,-1,Nose,-1,NoseType3
1,0.3,-1,Mouth,-1,MouthType3
1,0.3,-1,RightEar,-1,REarType3
1,0.3,-1,FaceShape,-1,FaceShapeType2
1,0.25,-1,LeftEye,-1,LEyeType4
1,0.2,-1,LeftEar,-1,LEarType2
1,0.4,-1,Mouth,-1,MouthType2
1,0.4,-1,RightEar,-1,REarType1
1,0.2,-1,LeftEye,-1,LEyeType2
1,0.3,-1,LeftEye,-1,LEyeType1
1,0.2,-1,RightEye,-1,REyeType2
2,0,-1,Eye,-1,LeftEye,-1,RightEye
1,0.3,-1,Mouth,-1,MouthType1
1,0.45,-1,Nose,-1,NoseType2
5,0,-1,Face,-1,FaceShape,-1,Eye,-1,Nose,-1,Mouth,-1,Ear
1,0.35,-1,RightEye,-1,REyeType1
1,0.35,-1,Nose,-1,NoseType1
1,0.25,-1,RightEye,-1,REyeType3
1,0.2,-1,RightEye,-1,REyeType4
1,0.25,-1,LeftEye,-1,LEyeType3
1,0.4,-1,LeftEar,-1,LEarType3
1,0.4,-1,LeftEar,-1,LEarType1
2,0,-1,Ear,-1,LeftEar,-1,RightEar
1,0.3,-1,RightEar,-1,REarType2
```

Figure 3.2: True face grammar in parameter learning



```
1,0.689379,-1,FaceShape,-1,FaceShapeType1
1,0.185557,-1,Nose,-1,NoseType3
1,0.307924,-1,Mouth,-1,MouthType3
1,0.330993,-1,RightEar,-1,REarType3
1,0.310621,-1,FaceShape,-1,FaceShapeType2
1,0.238956,-1,LeftEye,-1,LEyeType4
1,0.187563,-1,LeftEar,-1,LEarType2
1,0.419258,-1,Mouth,-1,MouthType2
1,0.411234,-1,RightEar,-1,REarType1
1,0.184739,-1,LeftEye,-1,LEyeType2
1,0.328313,-1,LeftEye,-1,LEyeType1
1,0.200803,-1,RightEye,-1,REyeType2
2,0,-1,Eye,-1,LeftEye,-1,RightEye
1,0.272818,-1,Mouth,-1,MouthType1
1,0.460381,-1,Nose,-1,NoseType2
5,0,-1,Face,-1,FaceShape,-1,Eye,-1,Nose,-1,Mouth,-1,Ear
1,0.355422,-1,RightEye,-1,REyeType1
1,0.354062,-1,Nose,-1,NoseType1
1,0.24498,-1,RightEye,-1,REyeType3
1,0.198795,-1,RightEye,-1,REyeType4
1,0.247992,-1,LeftEye,-1,LEyeType3
1,0.424273,-1,LeftEar,-1,LEarType3
1,0.388164,-1,LeftEar,-1,LEarType1
2,0,-1,Ear,-1,LeftEar,-1,RightEar
1,0.257773,-1,RightEar,-1,REarType2
```

Figure 3.3: Learned face grammar in parameter learning

CHAPTER 4

Structure Learning of the And-Or Graph

In the previous sections, we know the structure of the And-Or graph and only need to determine the parse tree or parameters of the potential functions. In this section we learn the structure of the And-Or graph behind the training data. In this section we use the 1D text data and another robotic commands dataset to illustrate the learning.

To learn the structure of the And-Or graph, we apply the algorithm that is introduced in [TPZ13]. In this paper, the notion of And-Or fragment is introduced. According to the paper, an And-Or fragment is a combination of and-node and or-node, with an and-node as the root and several or-node as its children. Each or-node can then have several children as leaf nodes. This And-Or fragment structure is better than fragments that only contain and-node and or-node as merely adding and-fragment to the grammar may be insufficient to cover the structure of the grammar while merely adding or-fragment decreases the posterior probability of the grammar and may need more expensive search algorithms [TPZ13].

In [TPZ13], an algorithm for adding And-Or fragment to the grammar piece by piece is introduced. In each iteration, a randomly chosen And-Or fragment with two or-nodes and two children for each or-node is chosen from data and then modified by deleting or adding or-node or leaf-node using its neighboring nodes in the data. The paper also introduces the notion of posterior gain, which is the product of likelihood gain and prior gain. These two notions are defined as:

likelihood gain: $\frac{P(X|G_{t+1})}{P(X|G_t)} = \frac{\prod_{i=1}^{n}\prod_{j=1}^{m_i}\|RD_i(a_{ij})\|^{\|RD_i(a_{ij})\|}}{\|RD\|^{n\|RD\|}} \frac{\prod_c(\sum_e CM[e,c])^{\sum_e CM[e,c]}}{\prod_{e,c} CM[e,c]^{CM[e,c]}}$
prior gain: $\frac{P(G_{t+1})}{P(G_t)} = e^{-\alpha(\|G_{t+1}\|-\|G_t\|)}$
The or-node weight of the And-Or fragment is set as:
$P(O_i \rightarrow a_{ij}) = \frac{\|RD_i(a_{ij})}{\|RD\|}$

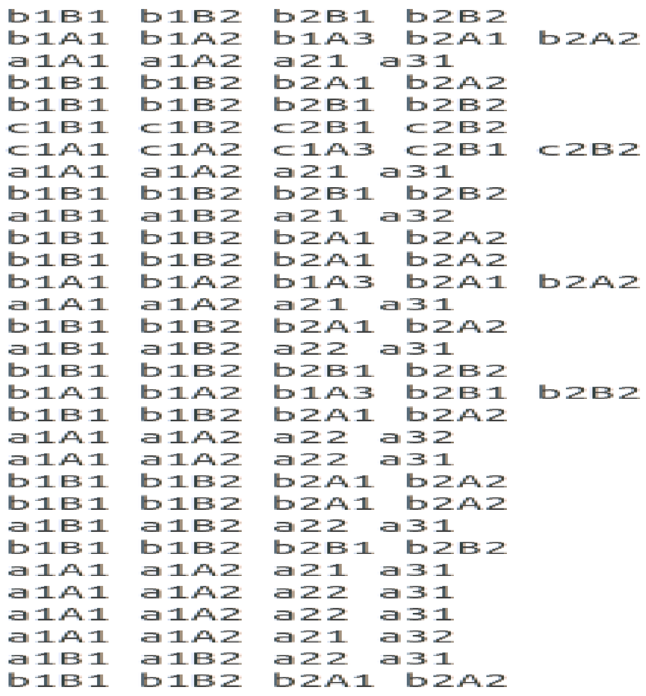
In the equation above, X is the observed data and G_t is the grammar at iteration t. n is the length of the configuration generated by the And-Or fragment, which is equal to the number of children or-node of the root node and m_i is the number of children for a or-node in the And-Or fragment. CM is a notion introduced in the paper as the context matrix for a selected And-Or fragment, where each entry represents the number of occurrence of one possible configuration of the And-Or fragment in that specific context in the training data. The context is defined as the rest of one training data with the parts that can be substituted by the And-Or fragment removed. RD here denotes the reductions we can make in the training data, which are the patterns in data that can be generated using the And-Or fragment. And $RD_i(a_{ij})$ are the reductions in which the i-th node reduced is a_{ij}. The last equation here shows that the And-Or fragment add in each iteration uses the weights that is proportional to the occurrences of the leaf nodes replaced in the training data, just like the way of setting or-node weights in section 2.2.

In this section, the form of training data for 1D text and robotic commands are shown in Figure 4.1 and Figure 4.2. The 1D text data is sampled from the same And-Or graph in section 2.1. We can implement the algorithm in [TPZ13], which is shown in Figure 4.3 with greedy search to learn the structure of the And-Or graph.

In practice, since we raise the number of reductions or configurations that satisfy certain conditions to the power of themselves, we may encounter values that are too large to compute. To solve this problem, we calculate the values in the log space. In each iteration, we generate in the And-Or graph new rules introduced by the And-

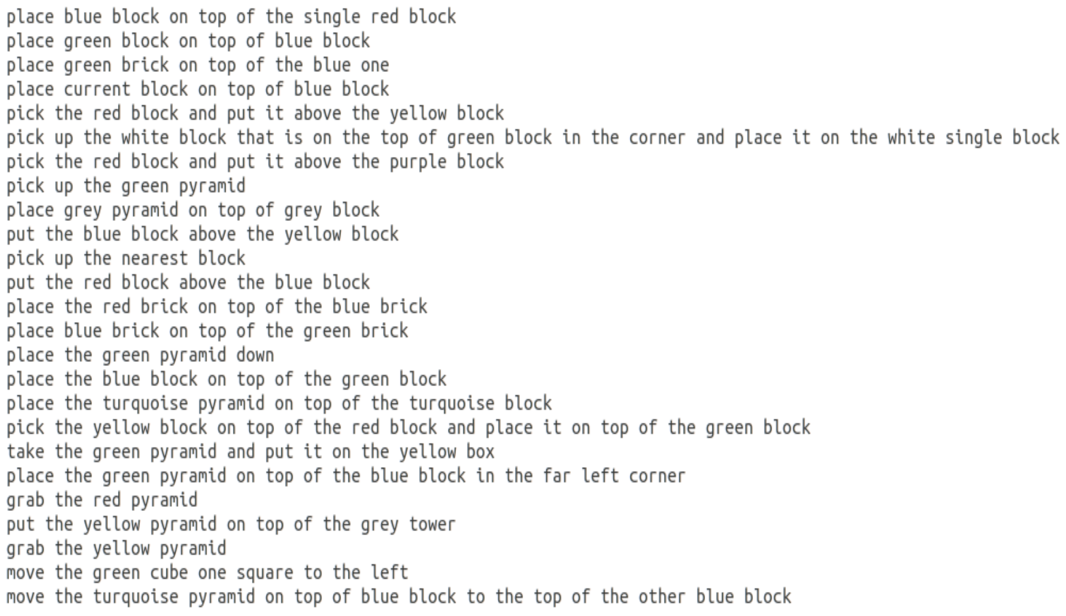
Or fragment. Also, since some parts of the original And-Or graph are replaced, some rules are removed from our grammar. Initially, the grammar is represented directly by all the training data observed. This can be very inefficient and we can not generate new configurations using the grammar. So although we can generate samples from the original dataset, the parsing accuracy of new testing data world be zero if the testing data doesn't appear in the training set. If the training goes well, we should be able to see that during the training process, the number of rules for representing the training data decreases. This is because more and more "data" are grouped together as "the same". Here "data" is actually the third level representation of the data in the And-Or graph if we regard the root or-node as the top level and the And-node of each different data as the second level. This data representation is then updated in each iteration. By "the same" we mean that although the data itself never change, as we update the And-Or graph, the third level representation for some of the data will become the same.

In conclusion, in this part we implemented the algorithm in [TPZ13] and applied it to a 1D text dataset and a robotic command dataset. After 50 iterations of learning, each iteration with 100 inner loops to choose new And-Or fragment, the generated data using the learned grammar for the 1D text and robotic commands is shown in Figure 4.4 and Figure 4.5. When using 2419 robotic commands sentences to learn and 794 robotic commands to test, 8.0506% of the unseen test robotic commands can be parsed correctly using Earley Parser [Ear70] in 50 iterations. The learned samples are not perfect and contain many fragments. To improve the result, we may need to use beam search as pointed out by the paper and use a larger expand limit to modify the randomly initialized And-Or fragment.



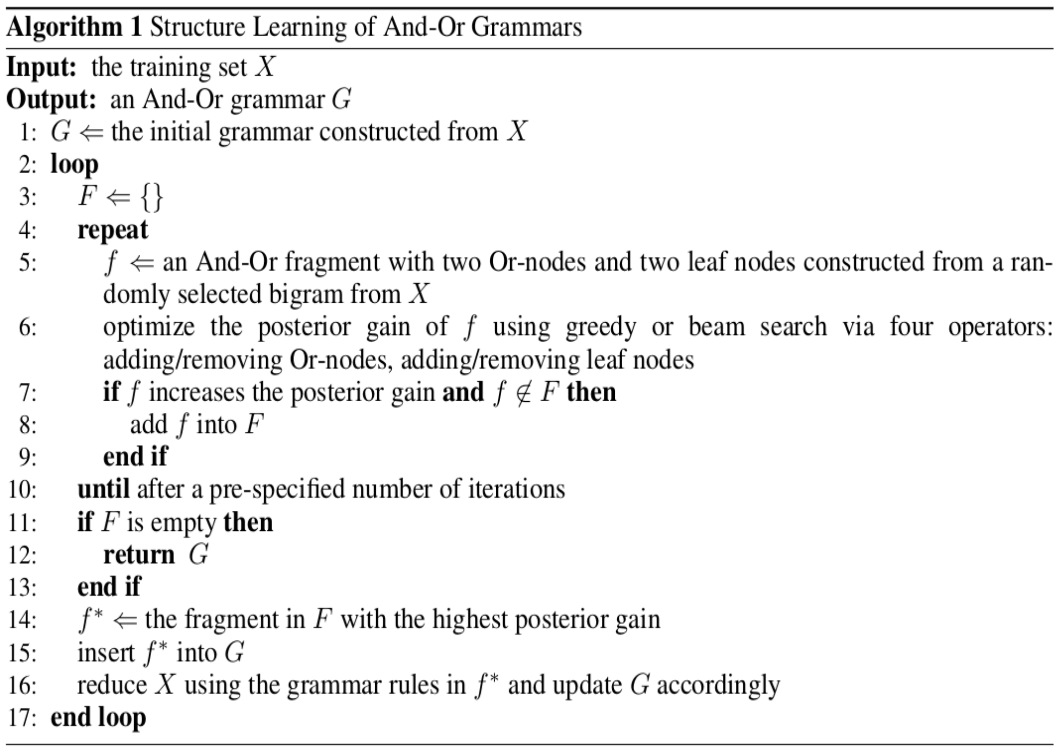
```
b1B1  b1B2  b2B1  b2B2
b1A1  b1A2  b1A3  b2A1  b2A2
a1A1  a1A2  a21  a31
b1B1  b1B2  b2A1  b2A2
b1B1  b1B2  b2B1  b2B2
c1B1  c1B2  c2B1  c2B2
c1A1  c1A2  c1A3  c2B1  c2B2
a1A1  a1A2  a21  a31
b1B1  b1B2  b2B1  b2B2
a1B1  a1B2  a21  a32
b1B1  b1B2  b2A1  b2A2
b1B1  b1B2  b2A1  b2A2
b1A1  b1A2  b1A3  b2A1  b2A2
a1A1  a1A2  a21  a31
b1B1  b1B2  b2A1  b2A2
a1B1  a1B2  a22  a31
b1B1  b1B2  b2B1  b2B2
b1A1  b1A2  b1A3  b2B1  b2B2
b1B1  b1B2  b2A1  b2A2
a1A1  a1A2  a22  a32
a1A1  a1A2  a22  a31
b1B1  b1B2  b2A1  b2A2
b1B1  b1B2  b2A1  b2A2
a1B1  a1B2  a22  a31
b1B1  b1B2  b2B1  b2B2
a1A1  a1A2  a21  a31
a1A1  a1A2  a22  a31
a1A1  a1A2  a22  a31
a1A1  a1A2  a21  a32
a1B1  a1B2  a22  a31
b1B1  b1B2  b2A1  b2A2
```

Figure 4.1: Structure learning training data: 1D text



```
place blue block on top of the single red block
place green block on top of blue block
place green brick on top of the blue one
place current block on top of blue block
pick the red block and put it above the yellow block
pick up the white block that is on the top of green block in the corner and place it on the white single block
pick the red block and put it above the purple block
pick up the green pyramid
place grey pyramid on top of grey block
put the blue block above the yellow block
pick up the nearest block
put the red block above the blue block
place the red brick on top of the blue brick
place blue brick on top of the green brick
place the green pyramid down
place the blue block on top of the green block
place the turquoise pyramid on top of the turquoise block
pick the yellow block on top of the red block and place it on top of the green block
take the green pyramid and put it on the yellow box
place the green pyramid on top of the blue block in the far left corner
grab the red pyramid
put the yellow pyramid on top of the grey tower
grab the yellow pyramid
move the green cube one square to the left
move the turquoise pyramid on top of blue block to the top of the other blue block
```

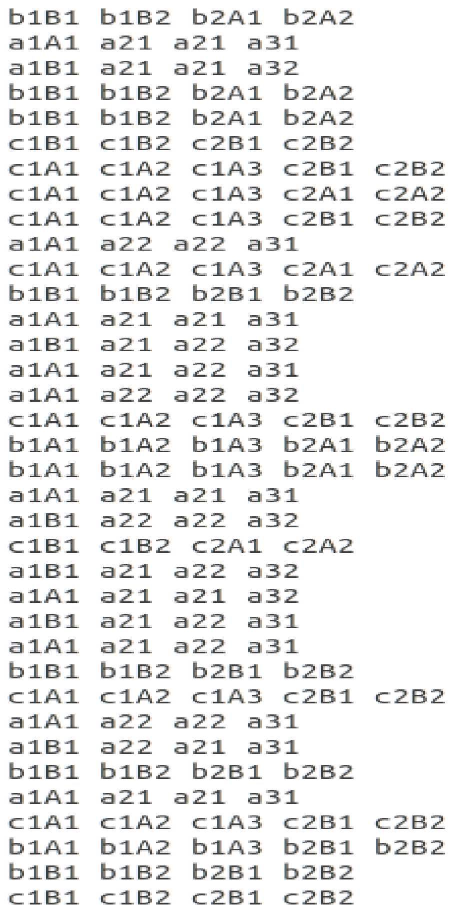
Figure 4.2: Structure learning training data: robotic commands



Algorithm 1 Structure Learning of And-Or Grammars

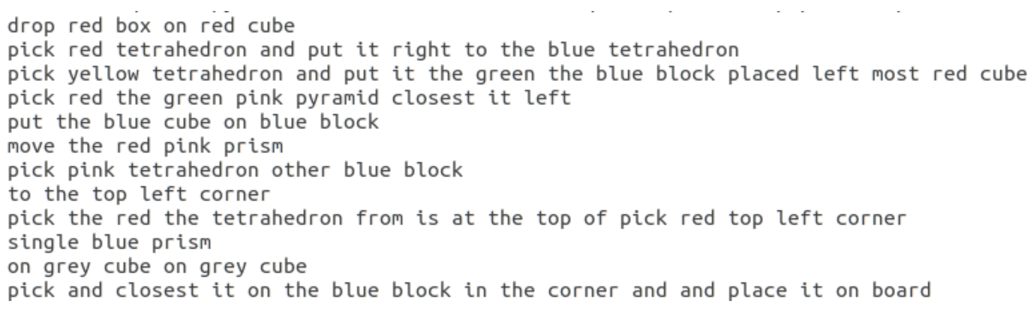
```
Input: the training set $X$
Output: an And-Or grammar $G$
 1: $G \Leftarrow$ the initial grammar constructed from $X$
 2: loop
 3:   $F \Leftarrow \{\}$
 4:   repeat
 5:     $f \Leftarrow$ an And-Or fragment with two Or-nodes and two leaf nodes constructed from a randomly selected bigram from $X$
 6:     optimize the posterior gain of $f$ using greedy or beam search via four operators: adding/removing Or-nodes, adding/removing leaf nodes
 7:     if $f$ increases the posterior gain and $f \notin F$ then
 8:       add $f$ into $F$
 9:     end if
10:   until after a pre-specified number of iterations
11:   if $F$ is empty then
12:     return $G$
13:   end if
14:   $f^* \Leftarrow$ the fragment in $F$ with the highest posterior gain
15:   insert $f^*$ into $G$
16:   reduce $X$ using the grammar rules in $f^*$ and update $G$ accordingly
17: end loop
```

Figure 4.3: The learning algorithm in [TPZ13]



```
b1B1 b1B2 b2A1 b2A2
a1A1 a21 a21 a31
a1B1 a21 a21 a32
b1B1 b1B2 b2A1 b2A2
b1B1 b1B2 b2A1 b2A2
c1B1 c1B2 c2B1 c2B2
c1A1 c1A2 c1A3 c2B1 c2B2
c1A1 c1A2 c1A3 c2A1 c2A2
c1A1 c1A2 c1A3 c2B1 c2B2
a1A1 a22 a22 a31
c1A1 c1A2 c1A3 c2A1 c2A2
b1B1 b1B2 b2B1 b2B2
a1A1 a21 a21 a31
a1B1 a21 a22 a32
a1A1 a21 a22 a31
a1A1 a22 a22 a32
c1A1 c1A2 c1A3 c2B1 c2B2
b1A1 b1A2 b1A3 b2A1 b2A2
b1A1 b1A2 b1A3 b2A1 b2A2
a1A1 a21 a21 a31
a1B1 a22 a22 a32
c1B1 c1B2 c2A1 c2A2
a1B1 a21 a22 a32
a1A1 a21 a21 a32
a1B1 a21 a22 a31
a1A1 a21 a22 a31
b1B1 b1B2 b2B1 b2B2
c1A1 c1A2 c1A3 c2B1 c2B2
a1A1 a22 a22 a31
a1B1 a22 a21 a31
b1B1 b1B2 b2B1 b2B2
a1A1 a21 a21 a31
c1A1 c1A2 c1A3 c2B1 c2B2
b1A1 b1A2 b1A3 b2B1 b2B2
b1B1 b1B2 b2B1 b2B2
c1B1 c1B2 c2B1 c2B2
```

Figure 4.4: Sample of 1D text generated by learned grammar



```
drop red box on red cube
pick red tetrahedron and put it right to the blue tetrahedron
pick yellow tetrahedron and put it the green the blue block placed left most red cube
pick red the green pink pyramid closest it left
put the blue cube on blue block
move the red pink prism
pick pink tetrahedron other blue block
to the top left corner
pick the red the tetrahedron from is at the top of pick red top left corner
single blue prism
on grey cube on grey cube
pick and closest it on the blue block in the corner and and place it on board
```

Figure 4.5: Sample of robotic commands generated by learned grammar

REFERENCES

[Cho57] Noam Chomsky. *Syntactic Structures.* Mouton and Co., The Hague, 1957.

[CXL06] Hong Chen, Zi Jian Xu, Zi Qiang Liu, and Song Chun Zhu. "Composite templates for cloth modeling and sketching." In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pp. 943–950. IEEE, 2006.

[Ear70] Jay Earley. "An efficient context-free parsing algorithm." In *Communications of the ACM*, 1970.

[HNF18] Tian Han, Erik Nijkamp, Xiaolin Fang, Mitch Hill, Song-Chun Zhu, and Ying Nian Wu. "Divergence Triangle for Joint Training of Generator Model, Energy-based Model, and Inference Model." *arXiv preprint arXiv:1812.10907*, 2018.

[TPZ13] Kewei Tu, Maria Pavlovskaia, and Song Chun Zhu. "Unsupervised structure learning of stochastic And-Or grammars." In *NIPS'13: Proceedings of the 26th International Conference on Neural Information Processing Systems on*, volume 1, pp. 1322–1330, 2013.