THE UNIVERSITY OF CHICAGO

GRAMMATICAL METHODS IN COMPUTER VISION

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
ERIC PURDY

CHICAGO, ILLINOIS
MARCH 2013

To my beloved wife

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

In computer vision, grammatical models are models that represent objects hierarchically as compositions of sub-objects. This allows us to specify rich object models in a standard Bayesian probabilistic framework. In this thesis, we formulate shape grammars, a probabilistic model of curve formation that allows for both continuous variation and structural variation. We derive an EM-based training algorithm for shape grammars. We demonstrate the effectiveness of shape grammars for modeling human silhouettes, and also demonstrate their effectiveness in classifying curves by shape. We also give a general method for heuristically speeding up a large class of dynamic programming algorithms. We provide a general framework for discussing coarse-to-fine search strategies, and provide proofs of correctness. Our method can also be used with inadmissible heuristics.

Finally, we give an algorithm for doing approximate context-free parsing of long strings in linear time. We define a notion of approximate parsing in terms of restricted families of decompositions, and construct small families which can approximate arbitrary parses.

CHAPTER 1
INTRODUCTION

We want to study grammatical methods in vision (also called compositional methods). Much past work has been done on this, including Amit and Trouvé [2007], Bienenstock et al. [1997], Fu [1986], Geman et al., Grenander and Miller [2007], Han and Zhu [2009], Jin and Geman [2006], Potter [1999], Tu et al. [2005], Zhu et al. [2009], Zhu and Mumford [2006], Felzenszwalb and McAllester [2010], but many fundamental questions remain unanswered.

Grammatical methods are characterized by the following:

- *Decomposing images hierarchically*, often in a semantically meaningful way. This is often referred to as *parsing* the image. Ideally, we would like an explanation for an entire scene. For example, in Figure 1.1, each image pixel belongs to at least one object (such as a car), and some objects are sub-parts of other objects (a wheel is part of a car).

- Part-based object models whose parts are other object models. For example, a model of a car would contain a model for a wheel, which could also be used to model wheels on their own.

- Models that contain *reusable parts*. For example, a model of a face could use a single model for both eyes. The reusable parts could also use themselves recursively, as is seen in fractal shapes.

- Modeling some object classes as mixtures of sub-class models. This is important because some conceptually meaningful classes such as "chair" contain wildly disparate elements (such as easy chairs and lawn chairs) that cannot be captured by a single model.

- Models that exhibit large amounts of structural variation. For example, trees of a single species will have wildly different shapes, which share some common properties.

In particular, object models that exhibit *choice* between different models of various object parts. These choices may be nested; for example, the number of branches in a tree is not fixed in advance, and neither is the number of sub-branches of each branch.

This is partly motivated by consideration of the human visual system:

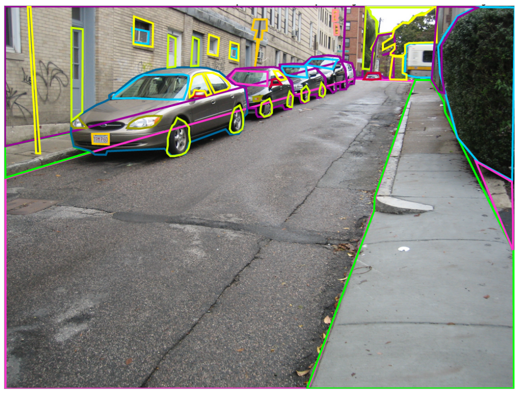
- Humans use context to resolve ambiguities (Bar [2004]). This means that we can only identify some objects by modeling their relationship to other objects. This can be seen in Amit and Trouvé [2007].
- Humans interpret some groupings of objects as a larger level object or activity, such as crowds of people or flocks of birds. Gestalt research demonstrates that perception has definite, repeatable grouping rules (Wertheimer [1938]).
- Humans seem to interpret whole scenes even when answering simpler visual questions, such as edge detection and segmentation. This can be seen in Figure 1.2, which shows an example from the Berkeley Segmentation Database (Martin et al. [2001]).

Grammatical methods are also motivated by theoretical considerations. They are a natural choice for modeling large structural variations (see Section 5.1). Grammatical models give a principled way to avoid hard decisions for low-level visual tasks (see Section 1.2). They allow strong models of background clutter (see Section 1.2). They allow whole scene parsing (see Section 1.2). They make it easy to integrate models from other domains into visual models (see Section 1.4.1). Finally, there are theoretical reasons why grammars may provide better generalization than other models.

1.1 Examples of Grammatical Approaches

1.1.1 Curve Grammars

We wish to build a model of closed curves in the plane. This is an important task because curves are the boundaries of objects, and we can use this fact to recognize some object

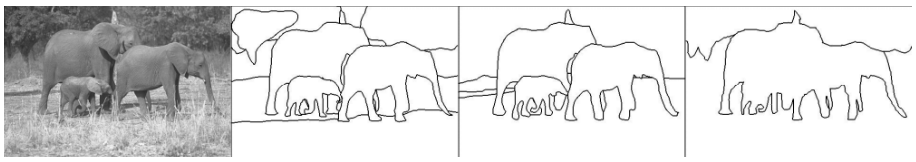


(a) An image with objects outlined.

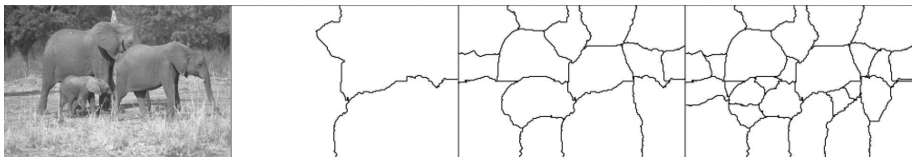
air conditioning	building
bus	car
door	headlight
license plate	mirror
plant	pole
road	sidewalk
sign	sky
traffic light	tree
wall	wheel
window	windshield

(b) Labels of the outlined objects.

Figure 1.1: An example of a parsed image. Adapted from the website of the LabelMe dataset (Russell et al. [2008]).



(a) Human Segmentation



(b) Machine Segmentation

Figure 1.2: Even on low-level visual tasks such as segmentation, humans give answers based on an interpretation of the whole scene. Figure adapted from Martin et al. [2001].

classes. The shape of a boundary is invariant to many photometric effects, in particular illumination (Canny [1986]).

1.1.2 Visual Chalkboard

As an example throughout this document, we discuss a system which would take images of a classroom chalkboard and attempt to parse them into lecture notes. This is an application with lots of noise. It is also an application where different levels of interpretation are required, since lectures can contain both sentences (which should be interpreted thoroughly, as text) and drawings (which could be interpreted partially as collections of lines, but which may contain unsummarizable elements which must be included verbatim).

In Section 1.4.1, we argue that grammars are a natural choice for such an application, because they make it possible to integrate statistical models from other domains in a straightforward and principled manner.

1.1.3 Visual Search Engine

Google image search is a nice and useful thing, but it relies partially on images being associated with relevant text on web pages. It would be nice to find raw or under-described images, and it would be nice to base search results more on the contents of the image. We might also submit images as queries, rather than text. The LabelMe dataset is a good challenge for this task.

In Section 1.3, we argue that hierarchical decomposition would allow the necessary rich understanding of relationships between objects.

1.1.4 Other Visual Grammars

Many well-performing vision algorithms can be thought of as special cases of grammatical algorithms. Some examples can be found in Amit and Trouvé [2007], Felzenszwalb and

Huttenlocher [2003], Felzenszwalb and McAllester [2010]. Recognizing these as a special case means that we may be able to improve upon this work by specifying richer models in some cases, or models that are better mathematically founded (and thus potentially trainable) in other cases. There are two tricks for turning a model into a grammar model:

- Mixture models are a special case of grammar models. If we have mixture components $M_1, \dots, M_k$ with mixture weights $p_1, \dots, p_k$, then we can build a grammar model M in which we have rules:

$$\begin{aligned} M &\to M_1 && (p_1) \\ &\to M_2 && (p_2) \\ &\dots \\ &\to M_k && (p_k) \end{aligned}$$

  The same is true of nearest neighbor models. This is exciting because mixture models and nearest neighbor models are often very powerful, but do not generalize well from a small amount of data.

- Deformable parts-based models are a special case of grammar models (Felzenszwalb and McAllester [2010]). Let $M(x)$ denote the hypothesis that model M appears at image location x. If we have part models $P_1, \dots, P_k$, then we can build a grammar model M which has the rules:

$$\begin{aligned} &M(x) \to P_1(x + \delta_1) + \cdots + P_k(x + \delta_k) \\ &P_i(x) \to P_i(x + \Delta) \\ &P_i(x) \to I(x_1 \pm w_i, x_2 \pm h_i) \end{aligned}$$

  The δ_i represent the ideal displacement of each part P_i from the object model. The model is deformable because the second kind of rule allows the parts to be randomly displaced. The probability of $P_i(x) \to P_i(x + \Delta)$ will depend on Δ. The third kind of rule gives the probability for placing a part, which can be thought of as the probability that a part P_i would produce the image data under it, $I(x_1 \pm w_i, x_2 \pm h_i)$.

In our model of curve grammars, the second kind of rule is given by the midpoint distribution $\mu_{X \to YZ}$, and the third kind of rule is trivial (see Chapter 2).

1.2 Grammatical Vision is Important

1.2.1 Soft Decisions

While human vision crucially relies on global context to resolve local ambiguity (Bar [2004]), computer vision algorithms often have a pipeline which makes hard low-level decisions about image interpretation, and then uses this output as input to higher-level analysis. Algorithms will be more accurate and less brittle if they can avoid making such hard decisions, as advocated in Amit and Trouvé [2007], Jin and Geman [2006], Felzenszwalb and Huttenlocher [2003].

For example, in the visual chalkboard, there will be various stray marks on the chalkboard. We would prefer not to filter these out with some sort of quality threshold, but instead mark them as possibilities, try to assemble an overall interpretation of the board, and then discount any stray marks that do not participate in the interpretation. This seems much more fruitful than filtering out stray marks, along with some genuine letters, and then having to be very forgiving of words actually missing some of their letters altogether.

This requires us to combine and resolve information at different levels. Grammatical methods provide us with powerful inference algorithms for determining the most likely decomposition of a scene under a given compositional model. Since important local ambiguities will lead to different global decompositions, this is exactly what is needed: the overall likelihood of the decomposition is a common currency that allows us to negotiate between fitting the local data well, and explaining the local data in a way that allows a good decomposition of the rest of the image.

1.2.2 Modeling Clutter with Object Sub-parts

We would like to build specific and accurate models of clutter. For instance, for the visual chalkboard, it would be helpful to have a model for stray chalk marks, rather than a model for arbitrary unexplained patches; otherwise we will be tempted to explain stray chalk marks as some letter, possibly a lower-case 'i'. If we try to set our threshold high enough that we don't do this, we might start labeling some genuine i's as background. If we instead have a model for chalk marks, we can explain stray chalk marks and i's as particular sorts of chalk marks, and differentiate them based on context and appearance.

Jin and Geman [2006] suggests modeling clutter in the background with sub-parts of the objects of interest. Since objects in the background are still objects, and are often related to the objects of interest, this might allow us to build a much stronger background model in many cases. In addition, by modeling clutter with sub-parts, we are less likely to hallucinate whole objects when we see sub-parts. Thus, it is especially important that we have a cheap way to explain clutter that closely resembles sub-parts of the objects of interest.

With such a system, we might even be able to ignore rather subtle clutter, such as some stray letters, or even words, from a previous lecture that was not completely erased. Clutter words would not be part of a line of text, and would thus be identifiable as clutter in the parsed output, where they would be excluded from the main body of text.

1.2.3 Whole Scene Parsing

It is useful to demand whole scene parses, since it avoids the need to fine-tune detection thresholds and decision boundaries (Amit and Trouvé [2007]). Consider the example of the visual chalkboard. Instead of having to set a filter on chalk marks to filter out stray chalk marks, we simply explain them and discount them, since they are not part of any larger structure, such as a word, that we find interesting.

1.3 Hierarchical Decomposition and Rich Description

It would be very useful if vision algorithms could achieve richer understanding of scenes, and produce richer descriptions of images. A rich understanding of a scene requires an understanding of the relationship between objects. Consider an image containing a person and two objects, where the person is pointing at one of the objects. This is an important piece of information, and it cannot easily be described by a list of the objects in the image.

Some scenes contain important objects that are nothing more than a particular grouping of other objects: a crowd is just a collection of people. Moreover, the nature of the collective object is determined partly by the relationship between its elements. A crowd and a marching band are two very different objects, but this difference cannot be expressed in a simple listing of objects. How can vision algorithms achieve this level of understanding, or even represent it?

One straightforward and general framework for rich description is a labeled hierarchical decomposition of a scene (Zhu and Mumford [2006]). This takes the form of a tree, where:

- The root node describes the entire image.
- Each node is labeled with the name of an object, and an area of the image described, which may be approximate.
- The children of a node describe sub-parts of the parent, and have areas inside the parent node's area.

We can explicitly encode such a description in an XML-like language. Since grammatical methods produce and work with such hierarchical decompositions, they give a natural model for such structures.

A marching band and a crowd would differ in the label assigned to the relevant node, but the children of those nodes would be similar, i.e., individual people.

In the example of the visual chalkboard, rich description would produce more useful output than simple text. For instance, in transcribing a series of boards as lecture notes, we would like to be able to label non-textual regions as figures and include them verbatim, or render them as a collection of lines. Figures could also include text labels, for which we would want to know both their relationship to the figure and their textual content; a hierarchical format such as XML would best represent the logical structure involved.

1.3.1 Training on rich annotations

We would also like to train grammars simultaneously on an image and a hand-made rich hierarchical description of that image. This ensures that our trained grammars will produce semantically meaningful decompositions of new images: the decomposition will have a structure similar to that produced by a human, and we will be able to transfer labels onto the nodes of the decomposition.

This will make it more feasible to output meaningful rich descriptions on a wide range of data. Stochastic grammatical methods allow us to put soft constraints on descriptions ("it is unlikely that a person will have an apple for a face"), which will be more flexible and less brittle than hard constraints ("a person's face can have eyes, nose, etc., but not fruit"). We can thus favor more realistic descriptions of scenes while still producing useful output on very unrealistic scenes (such as Magritte's "The Son of Man").

Note that we may still gain information from rich descriptions without a specified correspondence to particular parts of the image. Knowing the correct structure and guessing at the exact correspondence is no harder than guessing both the correct structure and the correspondence. Such descriptions would be less labor-intensive to produce, so we might be able to train on larger datasets.

In general, supervised learning is easier than unsupervised learning. In computational linguistics, this means that learning a grammar for natural language is much more tractable given samples of natural language that have been parsed. (These are called *bracketed sam-*

ples.) It is likely that training on rich annotations would also make learning visual grammars much easier.

1.3.2 Some Problems with Rich Description, and Solutions

For a number of reasons, dealing with rich descriptions is more complicated than dealing with simpler descriptions. Grammatical methods and hierarchical decomposition give ways around some of these problems. Some important issues are: Class/Subclass ambiguity and Questionable Parts.

First, it is worth noting that hierarchical decomposition degrades gracefully into a flat description, since we can always decompose the root node into a list of objects. Hierarchical decomposition presses, but does not force, us to explain how any two parts of our annotation are related, making for more useful description.

- Decomposition provides a reasonable and elegant solution to the Questionable Parts problem. David Marr explained it thus:

  > Is a nose an object? Is a head one? Is it still one if it is attached to a body? What about a man on horseback?
  >
  > These questions show that the difficulties in trying to formulate what should be [considered an object] are so great as to amount to philosophical problems. There really is no answer to them - all these things can be an object if you want to think of them that way, or they can be a part of a larger object...(Marr [1983])

  In any annotation system rich enough that we might simultaneously label a wheel and a car, or eyes and a face, in the same image, there is an arbitrary choice of how many things to label. Forcing these descriptions to be consistent is very difficult (Russell et al. [2008]). This is especially pronounced with agglomerations, like crowds of people. There is no point at which a group of people meaningfully becomes a “crowd”; two people are not considered a crowd, and one hundred people are considered a crowd, but it is impossible to draw a clear line between crowds and not-crowds.

Forcing consistency may even be counter-productive. If we must label every face in a crowd, then a crowd seen from a distance will have an enormous number of face labels, most of which are basically guesses. If we never label faces in a crowd, then our visual search engine may fail to retrieve images of a particular person when they are in a group, even if they are clearly visible.

Hierarchical decomposition describes a scene as a hierarchy of objects, and further decomposition of these objects may be optional; as long as we know something about the object's appearance, we don't have to demand that it be broken up into its constituent parts.

- Grammatical methods also naturally address the Class/Subclass Ambiguity problem: descriptions of an object can be general or specific, and the level of specificity is fairly arbitrary. It is clear that we want the query "dancer" in a visual search engine to return images of people dancing, but these same images should also be returned on the query "person".

  Grammatical methods model such ambiguity as OR nodes in an AND-OR structure (Zhu and Mumford [2006]), or by rules of the form

$$\begin{aligned} \text{CLASS} &\to \text{SUBCLASS}_1 \\ &\to \dots \\ &\to \text{SUBCLASS}_k. \end{aligned}$$

The LabelMe dataset uses a set of labels derived from WordNet (Fellbaum [1998]) that are related via class-subclass relationships. The maintainers of the dataset claim that it requires very little work to map the arbitrary labels provided by users into these more precise and formalized labels (Russell et al. [2008]). Given such techniques, the Class-Subclass ambiguity problem is probably not a fundamental barrier to rich description.

1.3.3 Rich Description and XML

The rich descriptions we have described are naturally represented in XML and similar languages, which yields some opportunities:

- XML is reasonably human-readable and human-writable. (Comparable formats like YAML are even more so.) This means that rich photo-tagging could be done by many people, and also that many users could benefit from a visual search engine that accepts structured queries. For example, photos inside of an image could be recursively described as such, allowing us to separate the images containing an actual movie star from those containing a poster depicting that movie star. As another example, having a notion of classes and subclasses would allow us to search for `BASS < FISH` and receive only pictures of bass fish, and not pictures of the instrument or of other fish.

- Existing technology such as XPath allows computer programs to do efficient and flexible searches on XML documents. This means that fairly complex image-sorting tasks could potentially be automated. This could be very good, because some image-sorting tasks such as content moderation are reported to be very psychologically damaging when performed by humans.

- One particular XML-based file format is the *scalable vector graphics* format (SVG). If we can learn rich visual grammars, we could hope to recover artistic information from images, so that we could approximate a drawing as a collection of strokes in fills in an SVG document.

Ultimately, we might hope to learn artistic concepts such as drawing style or font, which would greatly expand the power of graphics programs.

1.4 Grammars and Statistical Models

1.4.1 Statistical Modules

Grammatical methods offer a very powerful and general way to make vision algorithms more robust: if we can integrate different statistical models in a modular fashion, especially models trained in different contexts, then our system will be more robust than any single model.

Grammatical methods are well-suited to integrating any statistical model that depends on the qualities of image objects and the relationships between them. When we can map objects and relationships between domains (for example, mapping pictures of text to actual text), this allows us to import already-trained statistical models from very different domains.

Consider transcribing a lecture from the visual chalkboard. The system will better recover from misidentifying letters if it uses higher-level knowledge about the lecture's language and contents. In particular, we can build grammars that integrate such tried-and-true models as the n-gram model of letters (Manning and Schütze [1999]), the n-gram model of words (Manning and Schütze [1999]), a stochastic grammar model of phrase and sentence structure (Manning and Schütze [1999]), and topic models of word choice in the subject of the lecture (Blei et al. [2003]). All of these models can be trained on large corpora of text, rather than on smaller datasets of images.

1.4.2 Independence and the Poverty of Stimulus

Grammatical models are typically context-free, which is fundamentally about making independence assumptions. We argue that independence assumptions can increase the effective amount of training data you have.

Remark 1.4.1 (Context-freeness is an independence assumption). Grammatical methods in general are characterized by 1) a method for decomposing novel data in a hierarchical fashion, and 2) an explanation for each level of the hierarchy in terms of previously seen data. For example, in a standard CFG, we will have both a parse tree over a sentence, and a set of labels for the nodes of the parse tree. If the CFG has been automatically generated (i.e., no human has assigned semantic meaning to grammar elements), then the label's meaning will be derived solely from which parts of the training data have received that label.

For generic grammatical methods, we can define context-freeness by specifying that the contents of a node N (all nodes below N in the hierarchy) are independent of the context in which N appears (all nodes not below N in the hierarchy) given the data stored at N (its label and any attributes).

Remark 1.4.2 (Independence assumptions yield more effective data)**.** Independence assumptions let you effectively multiply the amount of data you have. Consider the following simple problem: try to classify 0-1 vectors into two classes given a bunch of labeled examples. Consider the two sort of extreme things we can do: if we assume that each coordinate of the vector is independent, we get a Naive Bayes classifier, where we basically learn what the distribution is over a given coordinate for each class. This can be done with a small number of training examples.

The other extreme is assuming that there is total dependence, that the vectors are just arbitrary elements of a set, and the individual coordinate values do not mean anything. Then the maximum likelihood classifier (Paranoid Bayes?) is given by seeing how many times a specific vector showed up in each class, and picking the class where it showed up more often. We would have to guess on any novel vector.

If the independence assumption is valid for the data, then the Naive Bayes classifier acts like the Paranoid Bayes classifier trained on a data set that is exponentially larger. (Specifically, for each class, we generate all possible vectors that can be made by taking the first coordinate of a random example from the class, then taking the second coordinate from an independently chosen random example from the class, etc.)

Even when the independence assumption does not apply, context-free grammars are useful. This can be seen in examining the classic nonsense sentence "Colorless green ideas sleep furiously". The supposed weakness of context-free grammars, that the context-free assumption is unrealistic, is actually a strength, because it allows us to parse and react to very unrealistic data.

1.5 Grammatical Vision is Difficult

Little concrete progress has been made with visual grammars. The general frameworks have few practical results, and people with practical results don't seem to have a general framework.

We are also hindered because the closest equivalent to linguistics is the cognitive science of vision, which is not as well-stocked with intermediate hypotheses and sanity checks. For example, in linguistics, utterances can be decomposed into phonemes. There is no agreement as to how a visual signal can be decomposed into canonical parts.

1.6 Grammar induction is difficult

Grammatical methods in vision are inspired by grammatical models like probabilistic context-free grammars in linguistics. In computation linguistics, the problem of learning a grammar from unlabeled examples (grammar induction) is very difficult.

There are many negative theoretical results (Lee [1996]), the most fundamental of them being Gold's Theorem:

Theorem 1.6.1 (Gold's Theorem)**.** *Consider an algorithm A trying to learn a language $L \in \Sigma^*$. A is given an infinite sequence of words $w_1, w_2, \dots \in L$ which contains every word in L at least once. After each w_i, A outputs some L_i. A is said to* learn L in the limit *if, for some n, $L = L_n = L_{n+1} = L_{n+2} = \dots$.*

Then, if a class of languages $\mathcal{L}$ contains all languages of finitely many strings, and at least one infinite language (in particular, if $\mathcal{L}$ is the set of context-free languages), no algorithm can learn all languages in $\mathcal{L}$ in the limit.

Note that this result applies to all algorithms, regardless of their complexity.

This is a fundamental theoretical barrier to grammar induction. There is some hope that we can defeat it by adopting a Bayesian approach, and only hoping to learn sufficiently simple grammars. This has been attempted several times, but no solution is known for the associated Bayesian learning problem, and a heuristic search strategy must be used (Cook [1976], Stolcke [1994], Nevill-Manning and Witten [1997]).

It is clear that the problem of learning context-free grammars applies directly to curve grammars, since part of our curve model actually is a PCFG (see Chapter 2). The learning problem might be easier if visual grammars can be simpler in structure than string grammars for natural languages. We don't know if this is the case.

1.7 Visual Grammars are hard for additional reasons

1.7.1 Comparisons are Always Inexact

The continuous nature of visual signals means that we will rarely if ever see exact agreement between two visual objects at the pixel level. Compare a well-defined linguistic object (such as a word) to a visual part such as an eye; an eye will look slightly different in every image because of natural variation and photometric effects.

Additionally, because language has well defined symbols, in the case of written language, or agreed-upon phonemes, in the case of spoken language, it is possible in computational linguistics to check whether two letters or two words are the same. Working with visual grammars is thus analogous to trying to solve the problems of computational linguistics simultaneously with the problems of speech recognition.

Additionally, hand-labeled training data is more readily available in computational linguistics. For example, there are datasets of sentences which come with ground-truth grammatical parses. Such information is generally unavailable in vision.

Without certain knowledge of correspondence between samples, generalization is difficult. We are required to infer correspondences between samples in an unsupervised manner. This generally leads to a unsupervised learning problem, such as clustering, or a correspondence problem.

1.7.2 Dealing with Scale

Computer vision usually tries to be scale invariant, since a small object closer to the camera naturally looks like a large object further from the camera. A doubled copy of an image is hopefully semantically the same as the original image.

We thus have two problems: when an object is too small, its internal structure may be lost. Generally, fine internal structural elements first become texture (discernible en masse but not individually) and then disappear completely. The larger object may still, however, be recognizable by shape.

When an object is too large, we are faced with the task of parsing internal structure of which we were previously unaware.

These difficulties can probably be dealt with. The first demands that we come up with a simple appearance model for every object, so that we can detect it without detecting any of its subparts. Decreased recall is probably OK, as smaller things are actually more difficult to see.

The second demands that every object be detectable at large scales, even those which have no sub-parts in our grammar. We can model this with simple infinitely recursive grammars; sufficiently simple ones can hopefully model almost anything, while basically preventing their being bias for either one of:

- A "larger" model with more internal structure

- A “smaller” model with less internal structure

1.7.3 *Plane grammars do not naturally admit efficient parsing*

Efficient algorithms for parsing strings with context-free grammars rely on dynamic programming, and ultimately on the fact that a string has only quadratically many contiguous substrings. The elements of visual grammars naturally live in the image plane, and thus do not have a linear order. A visual object need not even occupy a single contiguous region, in the case of occlusion. Therefore, a parsing algorithm might in principle have to consider any subset of the image elements as a node in the parse, leading to an exponential runtime.

1.8 Contributions

Our first contribution is the formulation of a class of shape grammars, a probabilistic model of curve formation that allows for both continuous variation and structural variation. We describe shape grammars in Chapter 2 and derive an EM-based training algorithm for shape grammars in Chapter 3. We demonstrate the effectiveness of shape grammars for modeling human silhouettes, and also demonstrate their effectiveness in classifying curves by shape.

Our second contribution is a general method for heuristically speeding up a large class of dynamic programming algorithms, given in Chapter 4. We provide a general framework for discussing coarse-to-fine search strategies, and provide proofs of correctness. Our method can also be used with inadmissible heuristics.

Our third contribution is local inference, a method for correcting parses with unintended reuse, given in Chapter 4. We define an energy minimization problem over low-level assertions that allows us to detect shapes in cluttered images.

Our fourth contribution is an algorithm for doing approximate context-free parsing of long strings in linear time, given in Chapter 6. We define a notion of approximate parsing in terms of restricted families of decompositions, and construct small families which can approximate arbitrary parses.

Chapters 2, 3, and 6 are based on joint work with my advisor, Pedro Felzenszwalb.

CHAPTER 2

GRAMMATICAL MODELS OF SHAPE

In this chapter, we describe a formalism for modeling shapes based on context-free grammars.

2.1 Introduction

We are building probabilistic models of shape. For now, we consider only models of shape that deal with the location in the plane of a fixed number of (ordered) landmarks $z_1, \ldots, z_n$, which in our case will define a curve outlining some shape. Some existing approaches to this problem are: Markov models (Basri et al. [1998], Grenander et al. [1991]), the Procrustes distance and its probabilistic counterpart, the Watson distribution (Dryden and Mardia [1998]), and active shape models (Cootes et al. [1995]).

We would like to prove that our models do as well or better in explaining the geometric variability of various shape classes. The easiest way to compare two generative probabilistic models is to examine samples from them, and subjectively assess their similarity to real data. This is inherently a very qualitative evaluation.

Our main goal in building shape models is to allow us to do inference on new curves. We can calculate a likelihood that a shape class generated a particular curve. We start by considering models defined by perturbations of a single input curve.

When comparing to other probabilistic models, we can evaluate them in a more quantitative way using the log-likelihood, which is defined as

$$H(\{x_1, \ldots, x_n\}, q) = -\sum_{i=1}^{n} \frac{1}{N} \log_2 q(x_i).$$

If $X = \{x_1, \ldots, x_n\}$ is unseen data, and a large enough sample to be statistically useful, then $H(X, q)$ is a good measure of how well a model explains unseen data. Smaller values of $H(X, q)$ suggest that q is a better model. One problematic aspect of the log-likelihood is that

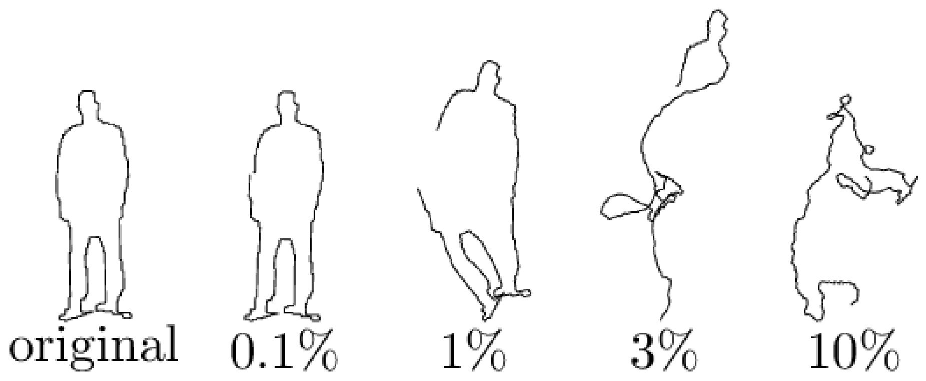


Figure 2.1: The problem of drift makes a Markov model unappealing. Random samples from this model are too similar locally and too dissimilar globally. These shapes were generated by changing each length l by a multiplicative factor of $1 + \mathcal{N}(0, \sigma)$, and changing each angle θ by adding $\pi \cdot \mathcal{N}(0, \sigma)$. Here σ is the value listed underneath the shape.

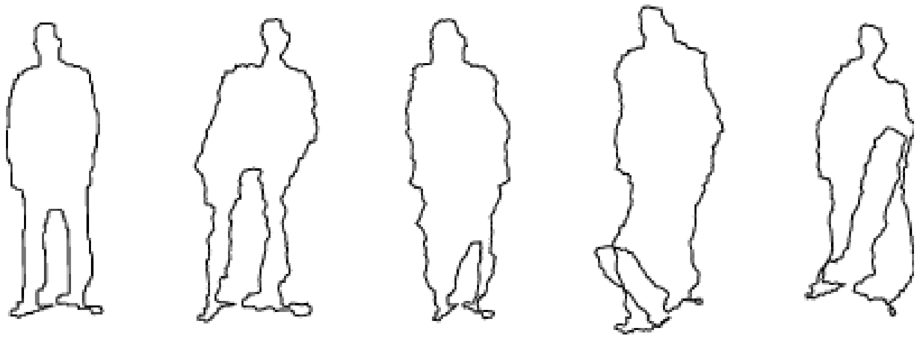


Figure 2.2: The shape on the left is the original, and the other curves have been produced by sampling from the hierarchical curve models of Felzenszwalb and Schwartz [2007] . The model produces curves which have more perceptual similarity than the Markov model.

it is only meaningful if q is a correctly normalized probability distribution. Often it is easier to compute $\widetilde{q}(x) = Z \cdot q(x)$ for some unknown but fixed constant Z. Due to the unsatisfactory nature of the competing models we consider, we defer quantitative measurement to Chapter 3.

The most straightforward model for defining a distribution over curve shapes is a Markov-type model. A curve is a sequence of line segments $\ell_1, \ldots, \ell_k$. We can describe a curve template by giving the length l_i of each ℓ_i, and the angle θ_i between ℓ_i and ℓ_{i+1}. If we perturb each l_i and θ_i slightly, we get a curve which differs from the original. Some samples from this are shown in Figure 2.1.

The major weakness of the Markov model is *drift*: the small perturbations will accumulate, and the overall shape of the curve will vary greatly. A straight line has some probability of curling into a tight spiral. Consider a shape like a hand: a hand has fingers that protrude. This means that there are two points (namely the two points where a finger meets the rest of the hand) far away in the curve that we always expect to be very close together. A Markov perturbation of the shape is likely to pull these points far apart.

To defeat this problem, *hierarchical curve models* were introduced in Felzenszwalb and Schwartz [2007]. There, the following curve model is given:

- Given a model curve C, decompose C hierarchically by repeatedly cutting it in half, in a balanced but otherwise arbitrary fashion.
- Suppose our first decomposition is $C = DE$. We perturb C into C' by first perturbing the midpoint of C slightly. We then rotate and scale the curves D and E to align the end of D and the beginning of E with the perturbed midpoint.
- We then recursively apply the same process to each subcurve D and E.

It is easy to see that this defeats the problem of drift, because the overall shape of the curve is determined in a constant number of perturbations of the template curve. If our perturbations are smaller for larger curves, then we will leave the overall shape very similar while allowing significant local variation. Some samples from this model are shown in Figure 2.2.

Hierarchical curve models have room for improvement, as can be seen in Figure 2.2. The variations do not respect the perceptual structure of the original curve; in particular, we do not see articulated parts being articulated.

In this chapter, we describe a grammatical reformulation of the work of Felzenszwalb and Schwartz [2007], which we hope will improve upon it in two ways. Firstly, we hope to allow for structural variation, which we argue is an important goal for computer vision in Section 5.1. Secondly, we give a generative probabilistic model of curves, which allows us to retrain the parameters of a grammar in a mathematically sound way, rather than optimizing many parameters in an ad-hoc manner (which will tend to be expensive and fragile).

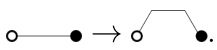
The rest of this chapter is structured as follows: in Section 2.2, we give an informal discussion of L-systems to motivate our models. In Section 2.3, we describe a method for sub-sampling curves in order to process them more efficiently. In Section 2.4, we lay out our grammar formalism. In Section 2.5, we show samples from several example grammars in order to demonstrate the expressiveness of our models. In Section 2.6, we describe how to use our models to parse novel curves. In Section 2.7, we describe our method of building models from a single example curve. In Section 2.8, we give models of triangle deformation, which are the building blocks of our models of shape. In Section 2.9, we give more details on how to set initial parameters of our models. Finally, in Section 2.10, we show output from our parsing algorithm.

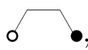
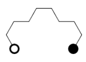
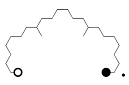
2.2 Motivating Example: L-Systems

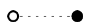
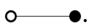
To motivate our discussion of curve grammars, we will first give an amusing example of such a grammar, which is inspired by L-systems. A Lindenmayer system, or L-system, is a parallel string rewriting system. L-systems were formulated to model the shape of plants and other organisms. They are defined on strings, which are then transformed into pictures using a "turtle language" akin to Logo. They are a simple and intuitive way to generate fractal images. Since our formalisms are different from L-systems, we will not discuss their details, which can be found in Prusinkiewicz [1990].

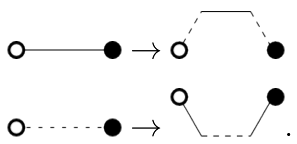
We will informally discuss curve rewriting systems, where we iteratively replace straight lines with curves composed of straight lines. We will represent our curves like this: $\circ\!\!-\!\!\!-\!\!\!-\!\!\bullet$, so that we remember which endpoint is which.

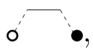
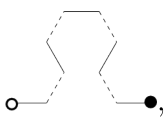
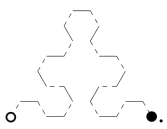
Consider the following system[1]:



Applying this rule a few times, we generate , ,  and .

We now wish to construct a more complicated system. We need to start distinguishing different kinds of curves, in this case  versus . Consider the following system[2]



If we start from , we successively generate , , and . Suprisingly, after a large number of iterations, a pattern arises that may be familiar: Sierpinski's triangle! After eight iterations, we have (filling in the dashed lines for simplicity) the curve shown in Figure 2.3.

These curves were generated by Inkscape's L-system function, which has a randomizing option. It is interesting to randomize the last example. In Figure 2.4, we see a randomized version C' obtained with fairly little noise (10% randomization in the length of line segments, and 5% randomization in the angles. These parameters only make sense in a more standard L-system framework). This has no global similarity at all to the other curve! Inkscape is using a Markov-like source of randomness, and little errors at the local level add up to huge changes at the global level. (This is exactly the issue shape-tree addresses in contrast to other deformable models.) If we are going to have a statistical model for perceptual similarity of curves, we will have to find a way to introduce long-range dependencies.

This brings us to our next section: probabilistic context-free grammars on curves. As a

1. generated by $A = A + A + A - -$ with angle 60° in Inkscape

2. generated by $A = B + A + B; B = A - B - A$ with angle 60° in Inkscape

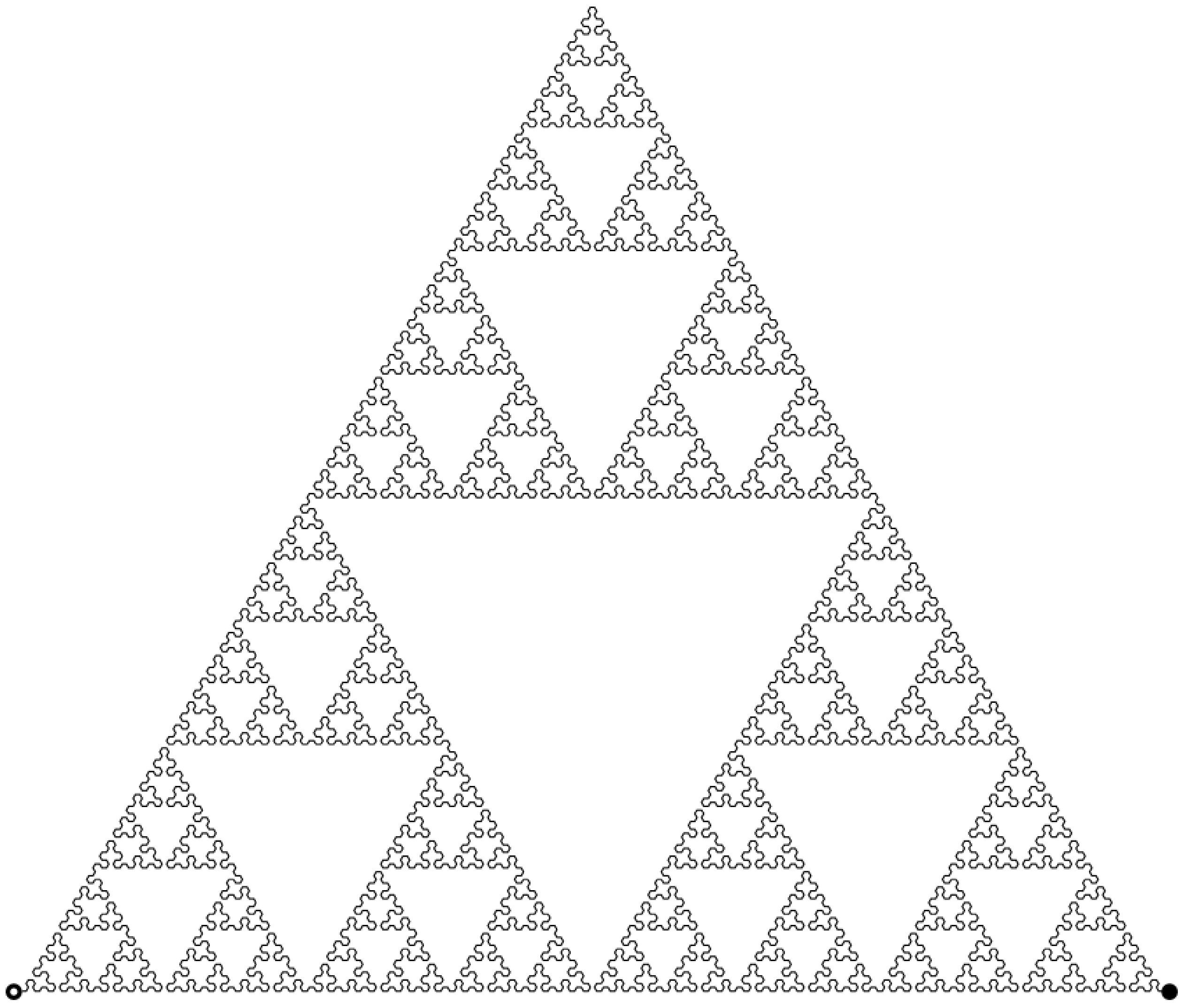


Figure 2.3: Sierpinski's triangle emerging from an L-system.

preview, we show several random perturbation of Sierpinski's triangle in Figure 2.5 that do retain overall similarity.

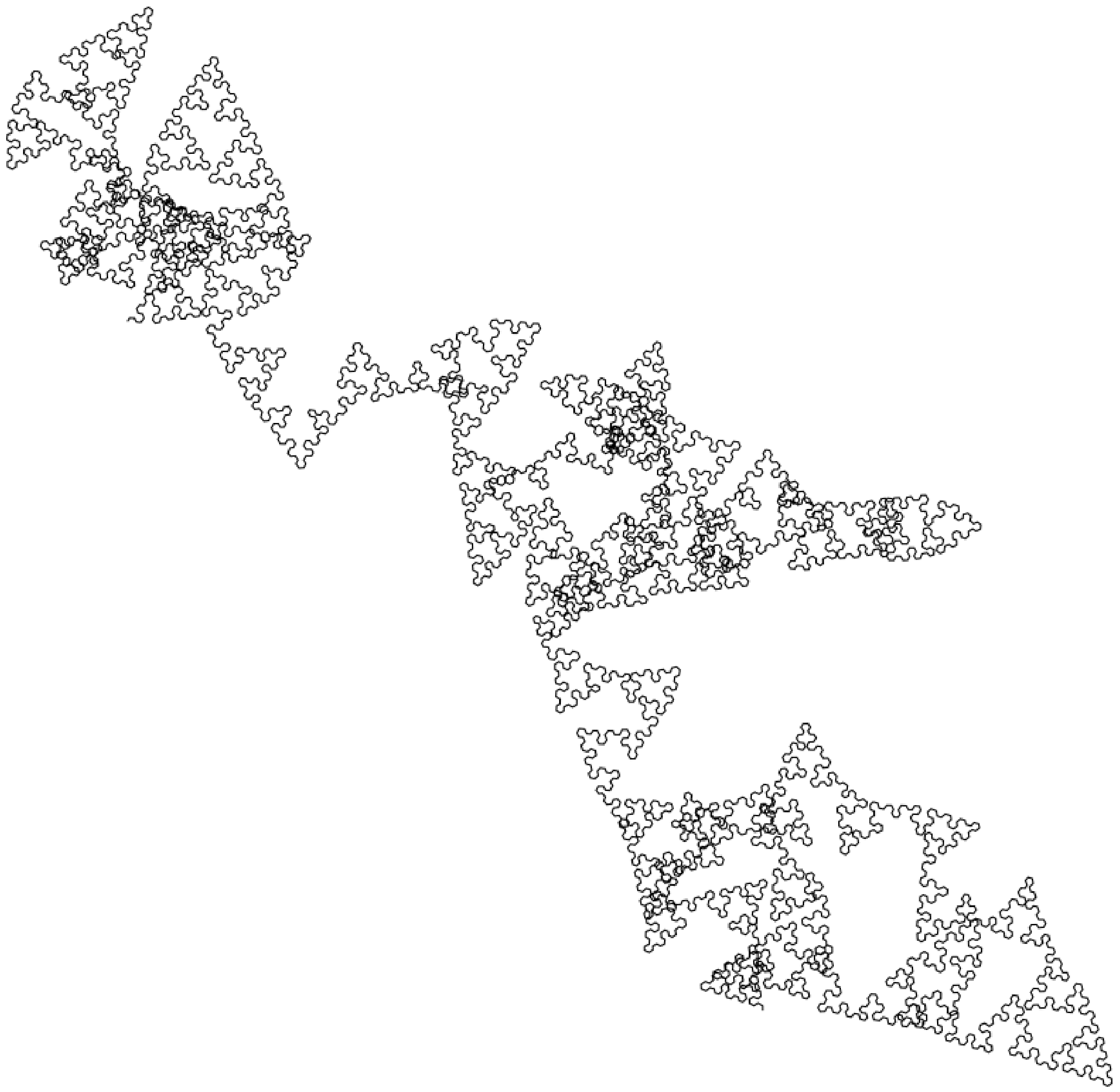


Figure 2.4: The curve from Figure 2.3, deformed randomly using a local model.

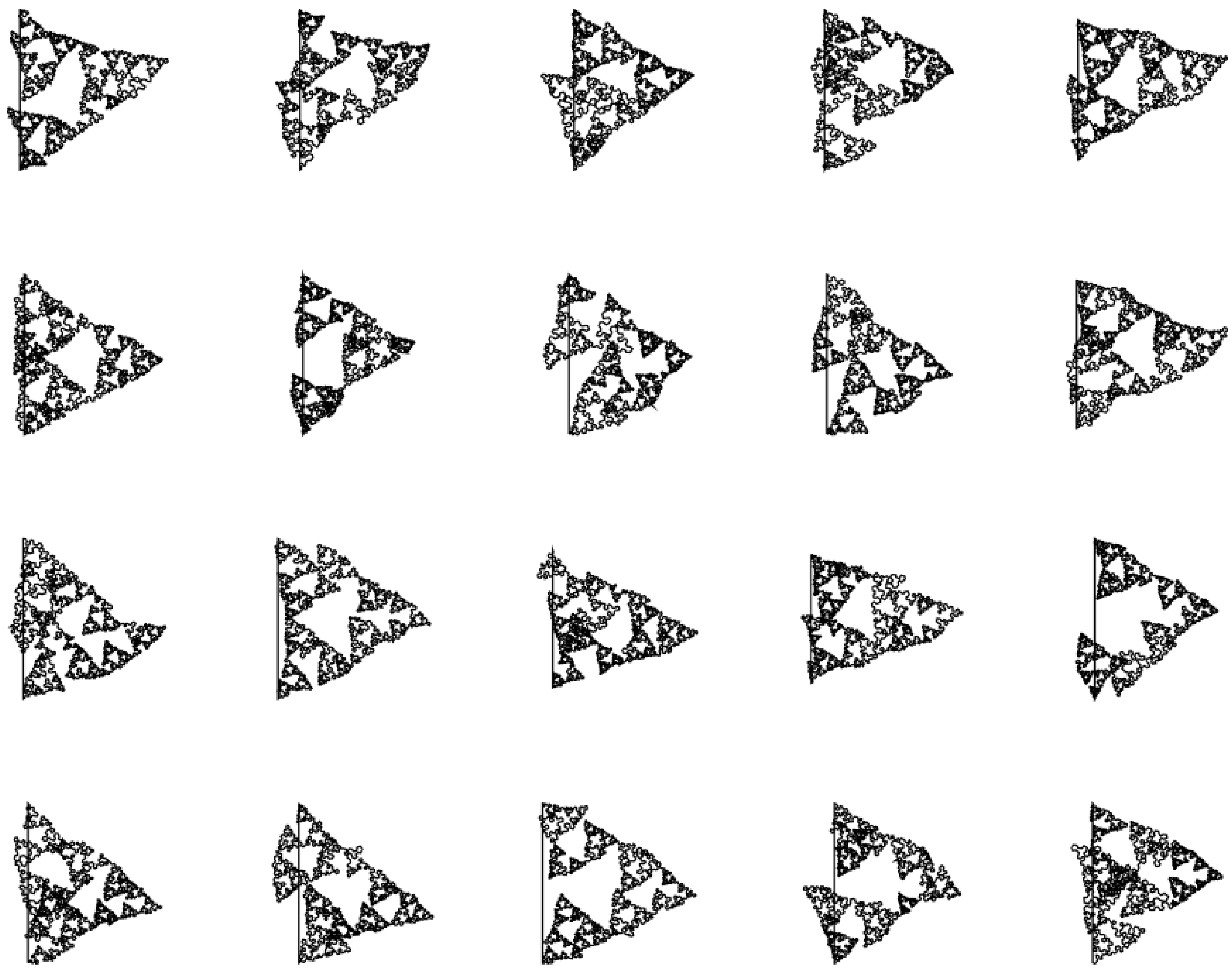


Figure 2.5: The curve from Figure 2.3, deformed randomly using our curve models.

2.3 Dealing with Curves

A continuous plane curve is a continuous function $C : [0, 1] \to \mathbb{R}^2$. C is closed if $C(0) = C(1)$. In order to handle curves computationally, we approximate them with discrete curves (referred to as "curves" hereafter). We create a curve by choosing a finite number of sample points $p_0, \ldots, p_n \in \mathbb{R}^2$, where $p_i = C(t_i)$ for some $0 \leq t_0 < t_1 < \ldots < t_n \leq 1$. A curve is closed if $p_0 = p_n$.

We can concatenate open curves C and D if the last point of C is the first point of D. We will denote this curve by CD.

We will denote an oriented line segment going from point p to point q by $\ell_{p,q}$. A curve will then have the form

$$\ell_{p_0,p_1}\ell_{p_1,p_2}\cdots\ell_{p_{n-1},p_n},$$

and p_0 will be equal to p_n iff the curve is closed. For a closed curve, this sequence is circular, and we consider any cyclic permutation of the line segments to be the same curve.

We will denote the length of a curve in segments by $|C|$. A curve C will have $|C| + 1$ sample points (counting $p_0 = p_n$ doubly if C is closed). We will denote the i-th sample point of C by $C[i]$, where i ranges from 0 to $|C|$.

2.3.1 Discretizing for Efficiency

Our inference and learning algorithms run in time that depends on the number of sample points in a curve. In particular, parsing takes time that is cubic in the number of sample points. We can therefore vastly speed up inference and learning by working with subsampled curves. In this section, we describe an algorithm that approximates the shape of a curve with another, coarser curve.

Let C be a curve with N points. We wish to produce a curve C' such that (a) C' approximates the shape of C, (b) $|C'| \approx L$, and (c) C' is sampled relatively uniformly from C. We do this by minimizing the objective function (2.1). The first term measures the total deviation between C and C', and the second term rewards relatively uniform sampling at the correct rate.

$$\operatorname{argmin}_{\{n_i\}} \sum_i \sum_{j=0}^{\lambda_i} \left\| p_{n_i+j} - \left(\frac{\lambda_i - j}{\lambda_i} p_{n_i} + \frac{j}{\lambda_i} p_{n_{i+1}} \right) \right\|^2 + \alpha \sum_i (\lambda_i - \widehat{\lambda})^2, \tag{2.1}$$

where λ_i is the length of the i-th segment and $\widehat{\lambda} = N/L$ is the "ideal" segment length. Here the n_i are the indices of the points selected to appear in the downsampled version of the curve, and the sum in the first term compares each point of the original curve to what we would interpolate it to be using the downsampled curve. If C is closed, then p_{n_i+j} wraps around: $p_{n_i+j} = p_{n_i+j \mod N}$.

This minimization can be done with a straightforward dynamic program, where we compute the minimum cost of approximating each subcurve of the curve. Note that we do not fix the number of sample points in advance, but instead use the number minimizing the cost. The results of the algorithm can be seen in Figure 2.6. Note that, while we lose fine detail, the resulting curves closely approximate the original curves.

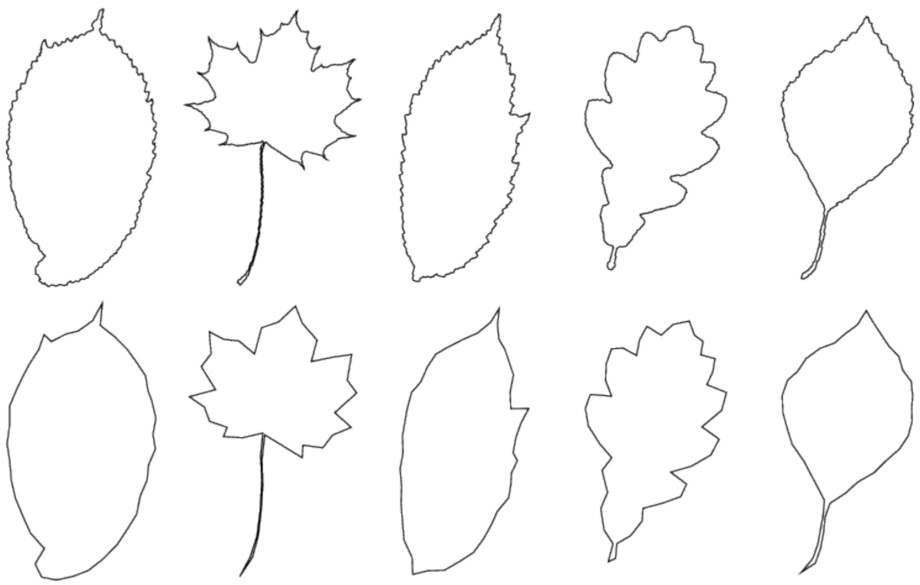


Figure 2.6: Subsampled curves. Here $\widehat{\lambda} = 40$

2.4 Stochastic Shape Grammars: A Generative Model for Curves

In this section, we define Probabilistic Context-Free Shape Grammars, a probabilistic model that allows us to generate random curves, parse curves to find their likelihood, and ultimately learn distributions over a class of curves.

Analogously to nonterminals in a traditional context-free grammar, we introduce *placed symbols*. A placed symbol is of the form $X_{p,q}$, where X is a member of a finite alphabet $\mathcal{N}$ or the special symbol ℓ, and p, q are points. $X_{p,q}$ represents an oriented curve of type X going from p to q. The path between the endpoints is unspecified.

We specify the type X so that different nonterminals can be related by the grammar. Recall $\circ\cdots\bullet$ and $\circ\!-\!\!-\!\bullet$ from the section on L-systems, which represented different kinds of curves because they were on the left-hand side of different productions. Therefore, X should be thought of as specifying a particular class of paths between any two endpoints. By itself, X is an *abstract symbol* that can be instantiated as a placed symbol between any p and q.

The special symbol ℓ denotes a line segment. This takes the place of terminals in traditional context-free grammars.

Definition 2.4.1. A *placed curvilinear form* (by analogy with a sentential form) is a sequence of placed symbols

$$\alpha^{(0)}_{p_0,p_1}\alpha^{(1)}_{p_1,p_2}\cdots\alpha^{(n-1)}_{p_{n-1},p_n},$$

where $\alpha \in \mathcal{N} \cup \{\ell\}$. As with curves, p_0 will be equal to p_n iff the curvilinear form is closed, and two closed curvilinear forms will be considered equivalent if they differ by a cyclic permutation.

An *abstract curvilinear form* is a sequence of abstract symbols (with no associated geometric information)

$$\alpha^{(0)}\alpha^{(1)}\cdots\alpha^{(n-1)}.$$

We will specify whether these are open or closed, since there is no other way to tell. Again, closed abstract curvilinear forms will be considered equivalent if they differ by a cyclic permutation.

We next introduce substitutions and substitution rules, which allow us to transform curvilinear forms, ultimately producing curves. We will perform substitutions of the form

$$X_{p,q} \to Y^{(1)}_{p,p_1} Y^{(2)}_{p_1,p_2} \cdots Y^{(k)}_{p_{k-1},q}.$$

Since $X_{p,q}$ represents an unspecified path from p to q, substitution simply gives a more specific route, in terms of which points we will visit in between (the p_i, which we will call the midpoint if there is only one of them, and *control points* otherwise) and what sort of path we will follow between these points (the $Y^{(i)}$).

In order to give a substitution rule for performing substitutions, we need to give

- An *abstract substitution rule* $X \to Y^{(1)} \cdots Y^{(k)}$.
- a rule for determining the p_i in terms of p and q.

In practice, applying a substitution rule is problematic because the p_i live in an infinite domain ($\mathbb{R}^2$), but we want to deal with curves that (1) live in a finite domain (the pixels of the image plane) and (2) are expected to exhibit a good deal of variation. Thus, we give a distribution $\mu_{X \to Y^{(1)} \ldots Y^{(k)}}(p_1, \ldots, p_{k-1}; p, q)$ over the p_i called the *control point distribution.* When there is only a single control point, we will call $\mu_{X \to YZ}(p_1; p, q)$ the *midpoint distribution.*

Definition 2.4.2. A probabilistic context-free shape grammar (PCFSG) is a tuple $\mathcal{G} = (\mathcal{N}, \mathcal{R}, \mathcal{S}, \ell, \mathcal{M}, \mathcal{X})$, where

- $\mathcal{N}$ is a set of abstract symbol types, which we will call *nonterminals*
- $\mathcal{R}$ is a set of abstract substitution rules, with $\mathcal{R}(X)$ being the set of rules in $\mathcal{R}$ with X on the left-hand side. The rules are of the form $X \to Y_1 \ldots Y_k$, with the Y_i in $\mathcal{N} \cup \{\ell\}$.
- $\mathcal{S}$ is the *starting set* of abstract curvilinear forms

- ℓ is a special curve type representing a line segment
- $\mathcal{X} = \{\rho_X \mid X \in \mathcal{N}\} \cup \{\rho_{\mathcal{S}}\}$, where ρ_X is a probability distribution over $\mathcal{R}(X)$, and $\rho_{\mathcal{S}}$ is a distribution over $\mathcal{S}$
- $\mathcal{M} = \{\mu_{X \to Y^{(1)} \ldots Y^{(k)}} \mid (X \to Y^{(1)} \cdots Y^{(k)}) \in \mathcal{R}\}$ is a set of control-point distributions.

It is worth noting that $\mathcal{G}$ has a corresponding *abstract grammar* $\mathcal{G}_{abs} = (\mathcal{N}, \mathcal{R}, \mathcal{S}, \{\ell\}, \mathcal{X})$ which is just a traditional PCFG. $\mathcal{G}_{abs}$ is an odd PCFG because it only generates strings of the symbol ℓ; nevertheless, many properties of $\mathcal{G}$ are actually properties of $\mathcal{G}_{abs}$.

For a grammar that generates open curves, we can assume that $\mathcal{S}$ has a single curvilinear form S, and we will call S the start symbol. We sample from such a PCFSG by starting with the curvilinear form $S_{p,q}$ for arbitrary p and q. While our curvilinear form contains a placed symbol $X_{p',q'}$ such that $X \neq \ell$, we pick a random substitution rule $X \to Y^{(1)} \ldots Y^{(k)}$ according to ρ_X, and pick random control points $p_1, \ldots, p_{k-1}$ according to $\mu_{X \to Y^{(1)} \ldots Y^{(k)}}(p_1, \ldots, p_{k-1}; p', q')$. We then replace the placed symbol $X_{p',q'}$ with the curvilinear form $Y^{(1)}_{p',p_1} \ldots Y^{(k)}_{p_{k-1},q'}$. We will disallow substitutions in which any two control points p_i, p_j are equal, or in which any control point p_i is equal to either of the endpoints p' or q'. This prevents us from having degenerate placed symbols $X_{p,p}$.

There is a slight difficulty here in that we usually cannot define the control-point distribution if p and q are equal. This is important, since we are mainly interested in closed curves, so we would like to start with $S_{p,p}$ for some arbitrary p. In this case, our set $\mathcal{S}$ of starting forms will contain abstract curvilinear forms of length two, which are understood to be closed. If $XY \in \mathcal{S}$ is our starting form, we start with the curvilinear form $X_{p,q}Y_{q,p}$ for arbitrary p, q. We choose a random such curvilinear form according to $\rho_{\mathcal{S}}$, and then continue as before.

2.4.1 Restricted Classes of Shape Grammars

In this section, we define some restricted classes of PCFSG's. Our restrictions are only on the abstract part of the grammar, so we are just specifying restricted classes of traditional grammars.

For simplicity and efficiency, we will usually restrict our abstract grammars to be in Chomsky Normal Form, in which abstract substitution rules can only be of two forms:

- Binary rules of the form $X \to YZ$.
- Lexical rules of the form $X \to \ell$.

From now on, we will assume that PCFSG's are in Chomsky Normal Form. Accordingly, we will speak of midpoints, rather than control points. It is important to note that PCFSG's differ from standard PCFG's in that not all PCFSG's are equivalent to a PCFSG in Chomsky Normal Form. This is because not all control point distributions can be realized as a product of midpoint distributions. For instance, suppose we have a rule $A \to BCD$, which in the PCFG setting could be replaced with rules $A \to BX, X \to CD$. We could have a control point distribution $\mu_{A\to BCD}(p_1, p_2; p, q)$ in which p_1, p_2 were on a random circle going through p and q. If we try to replicate such a control point distribution with midpoint distributions $\mu_{A\to BX}(p_1; p, q), \mu_{X\to CD}(p_2; p_1, q)$, we cannot, because the distribution $\mu_{X\to CD}(p_2; p_1, q)$ can only depend on the two points p_1 and q, and that is not sufficient to specify a unique circle.

2.5 Example Grammars

Here we demonstrate the expressiveness of grammatical curve models with some simple examples. We have built three grammars by hand, by taking the curves shown in Figure 2.8, and specifying decompositions of them. We show the rules of one of the grammars in Figure 2.7. We show samples from the grammars in Figures 2.9, 2.10, and 2.11. In particular, we show that a hand-built grammar can exhibit interesting variation, such as:

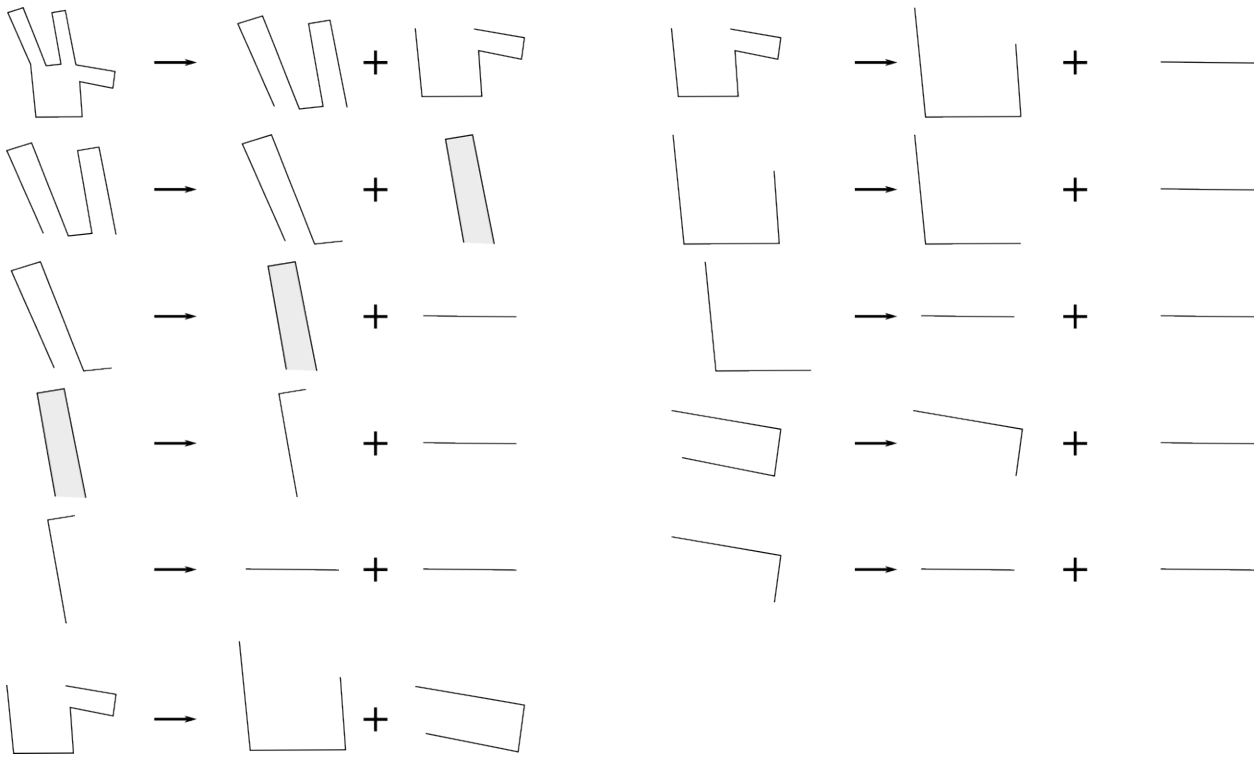


Figure 2.7: The rules for one of our hand-built grammars. The shaded part illustrates a reusable part because it occurs on the right hand side of two different rules.

- Small geometric deformation (Figure 2.9)
- Articulating parts of an object (fingers on the simplified hand in Figure 2.10)
- The presence or absence of a part (thumb on the simplified hand in Figure 2.10). This is achieved by giving the nonterminal which generates the thumb the rule $X_{thumb} \to \ell$.
- A choice between two different variations on a part (pants vs. skirt in 2.11). This is achieved by having rules $X_{lower} \to Y_{skirt} Z_{skirt}$ and $X_{lower} \to Y_{pants} Z_{pants}$, and having Y_{skirt}, Z_{skirt} based on a decomposition of the skirt-wearing shape, and Y_{pants}, Z_{pants} based on a decomposition of the pants-wearing shape.
- Shared parts that occur in different contexts (fingers on the simplified hand in Figure 2.10)

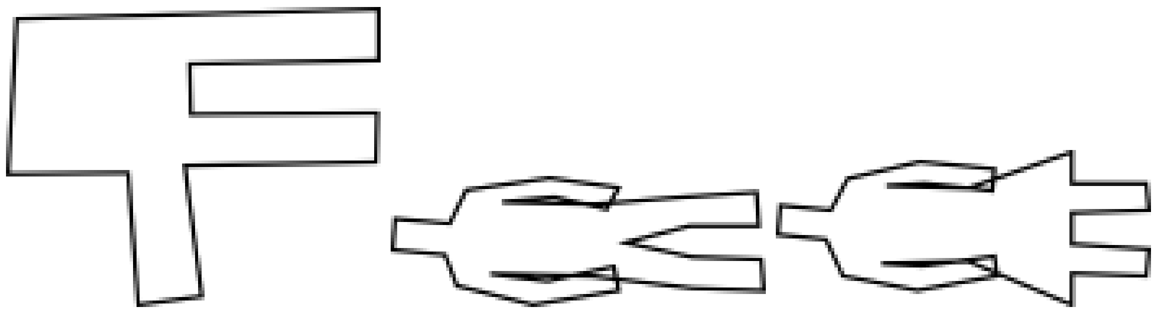


Figure 2.8: The curves used to initialize our hand-built grammars.

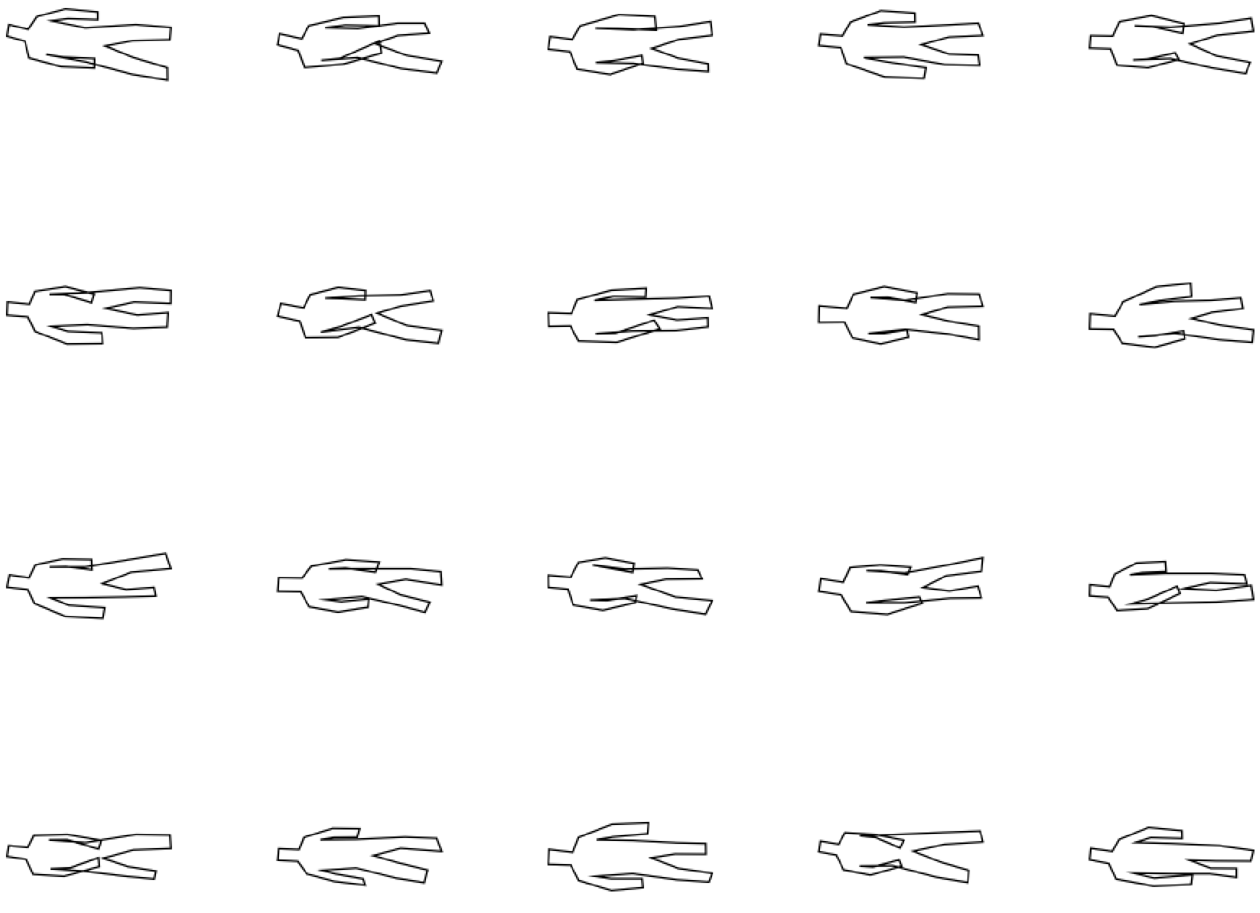


Figure 2.9: Samples from a grammar showing slight geometric deformation.

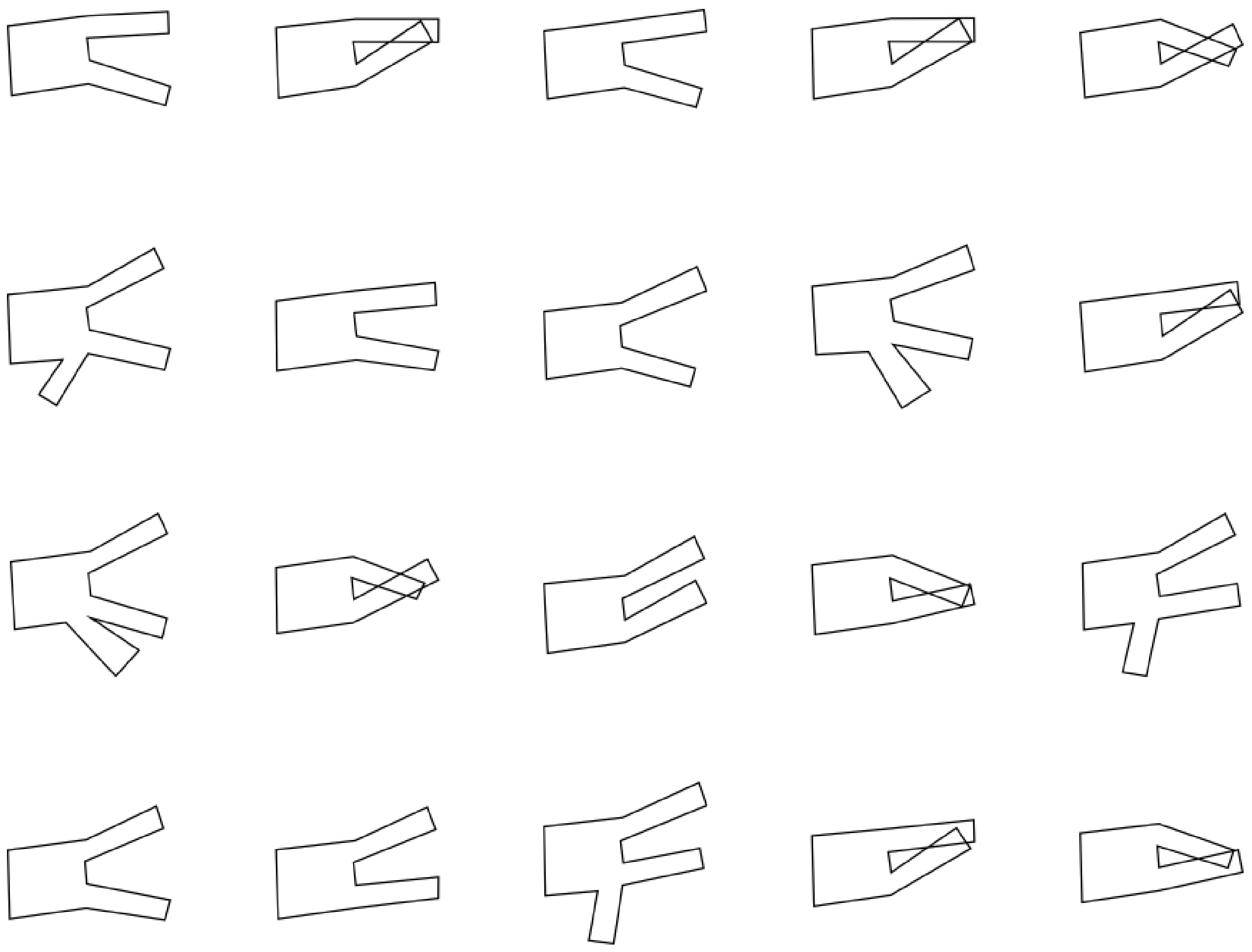


Figure 2.10: Samples from a grammar showing articulating parts, reusable parts, and a part that can be absent.

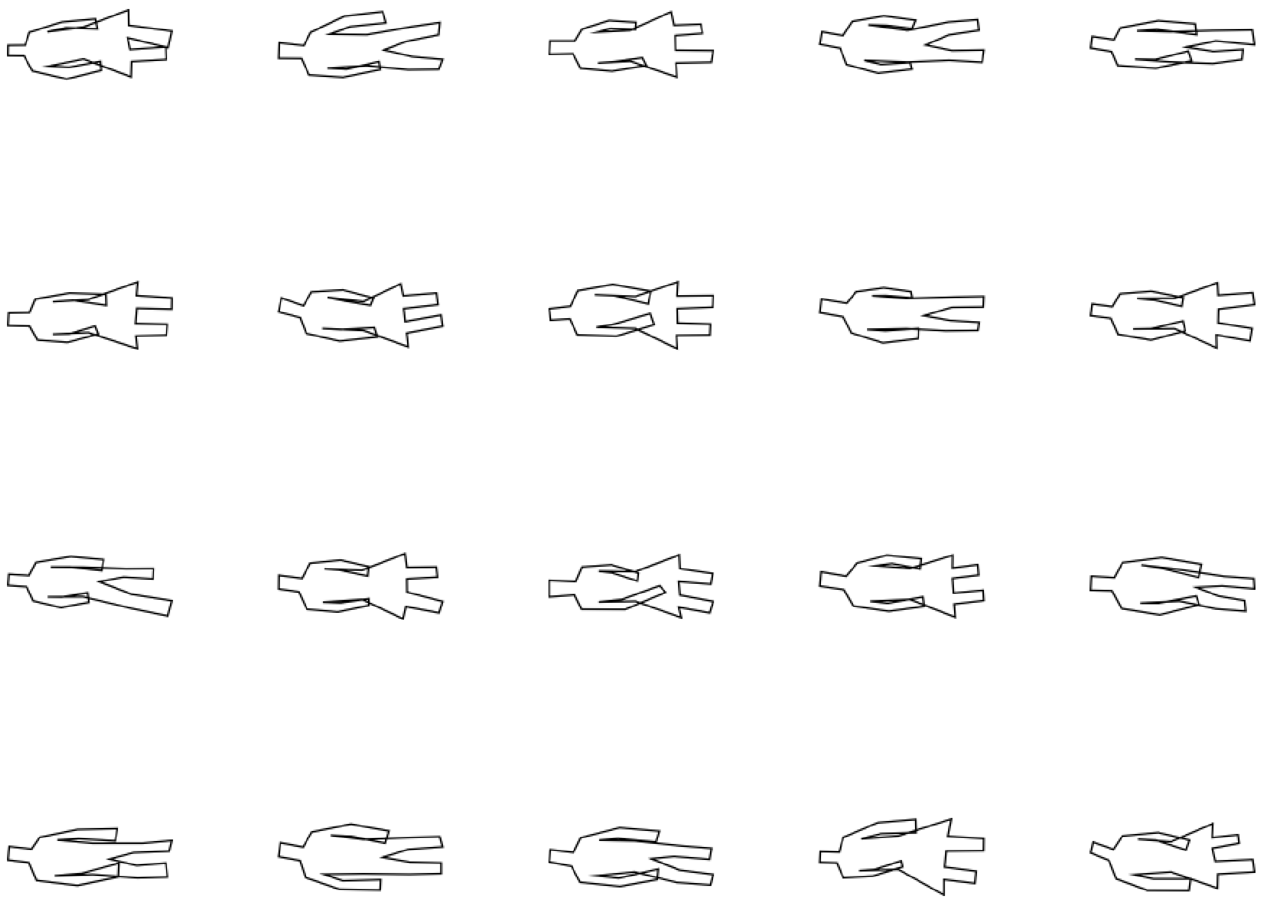


Figure 2.11: Samples from a grammar showing a choice between two different variants on a part, corresponding to a choice between pants and skirt.

2.6 Parse trees and Inference

Given a placed symbol $X_{p,q}$, the PCFSG defines a distribution over concrete substitution rules that can be applied:

$$P_{sub}(\langle X_{p,q} \to Y_{p,m} Z_{m,q}\rangle; X_{p,q}) = \rho_X([X \to YZ]) \cdot \mu_{X \to YZ}(m; p, q)$$
$$P_{sub}(\langle X_{p,q} \to \ell_{p,q}\rangle; X_{p,q}) = \rho_X([X \to \ell]).$$

We can then define $P(C; \mathcal{G}, p, q)$ to be the probability that we arrive at C by starting with $S_{p,q}$ and repeatedly applying a random concrete substitution rule according to P_{sub}. For a fixed grammar $\mathcal{G}$ and a given curve C with points $C[0], \dots, C[n]$, there are potentially many ways for this sampling process to produce C. In this section, we formalize this by defining parse trees.

For the sake of simplicity, we will assume in this section that all curves are open, unless otherwise specified. We deal with the technicalities of closed curves in Subsection 2.6.5.

2.6.1 Preliminaries

Definition 2.6.1. We define the set $\mathcal{F}_{\mathcal{G}}$ to be the set of all strings of placed symbols $X^{(1)}_{p_0,p_1} X^{(2)}_{p_1,p_2} \dots X^{(k)}_{p_{k-1},p_k}$ with $X^{(i)} \in \mathcal{N}$, where neighboring placed symbols $X^{(i)}_{p_{i-1},p_i} X^{(i+1)}_{p_i,p_{i+1}}$ are constrained to share the point p_i.

Let C be a curve of length n, with points $C[0]$ through $C[n]$. Since C is fixed, we will abuse notation throughout this section by writing placed symbols $X_{C[i],C[j]}$ as X_{ij}.

We will write a binary tree with root node v, left subtree T_1, and right subtree T_2 as $\frac{\qquad v \qquad}{T_1 \quad | \quad T_2}$.

Definition 2.6.2. Let T_1, T_2 be labeled binary trees, where u is a leaf node of T_1 and v is the root node of T_2. We define a new tree $T_1 \triangle^{\not{u}}_{v} T_2$ to be the tree resulting from deleting u and attaching T_2 in its place.

In order to differentiate between abstract and concrete substitutions we will write the former as $[X \to YZ]$ or $[X \to \ell]$ and the latter as $\langle X_{ik} \to Y_{ij} Z_{jk} \rangle$ or $\langle X_{ii+1} \to \ell_{ii+1} \rangle$.

2.6.2 Parse Trees

Definition 2.6.3. A $\mathcal{G}$-parse tree is a binary tree T that specifies a set of concrete substitution rules that take a placed symbol X_{ij} to some placed curvilinear form λ.

T has three kinds of nodes:

- Unexpanded nodes, which are labeled with a placed symbol X_{ij}, where $0 \le i < j \le n$ and $X \in \mathcal{N}$. Unexpanded nodes are always leaves.

- Lexical nodes, which are labeled with a concrete substitution rule

  $$\langle X_{ii+1} \to \ell_{ii+1} \rangle,$$

  where $[X \to \ell] \in \mathcal{R}(X)$, $X \in \mathcal{N}, 0 \le i < n$. Lexical nodes are always leaves.

- Binary nodes, which are labeled with a concrete substitution rule

  $$\langle X_{ik} \to Y_{ij} Z_{jk} \rangle,$$

  where $X, Y, Z \in \mathcal{N}, [X \to YZ] \in \mathcal{R}(X)$, and $0 \le i < j < k \le n$. Binary nodes always have two children. The first must be either of the form $\langle Y_{ij} \to \lambda \rangle$ or of the form Y_{ij}. The second must be either of the form $\langle Z_{jk} \to \lambda \rangle$ or of the form Z_{jk}.

For brevity, we will refer to $\mathcal{G}$-parse trees simply as parse trees.

To simplify notation, we will define the symbol of a node (denoted $sym(v)$) as:

- $sym(X_{ij}) = X_{ij}$

- $sym(\langle X_{ii+1} \to \ell_{ii+1} \rangle) = X_{ii+1}$

- $sym(\langle X_{ik} \to Y_{ij} Z_{jk} \rangle) = X_{ik}$

We have defined parse trees to represent derivations according to the substitution rules of $\mathcal{G}$. For $X \in \mathcal{N}, 0 \leq i, j \leq n$, the set of all $\mathcal{G}$-parse trees T such that $sym(root(T)) = X_{ij}$ is in one-to-one correspondence with the set of all derivations of placed curvilinear forms from the placed symbol X_{ij}. Unexpanded nodes correspond to incomplete derivations, while lexical and binary nodes correspond to applications of lexical and binary substitution rules. The constraints on the children of binary nodes require that our derivation then operate on the placed symbols produced by the substitution rule.

Definition 2.6.4. The *weight* of a parse tree T is defined to be

$$W_{\mathcal{G}}(T) = \prod_{\langle X_{ij} \to \lambda \rangle \in T} P_{sub}(\langle X_{ij} \to \lambda \rangle).$$

Note that this product omits the unexpanded nodes of T.

The weight of a parse tree multiplies the probability of each concrete substitution in the parse tree. We call it a weight rather than a probability because two different parse trees are not always mutually exclusive events (in particular, if one is a subset of the other), and thus $W_{\mathcal{G}}(T)$ does not sum to one over the set of all parse trees. Trees with unexpanded nodes correspond to sets of trees, and the weight of the tree is the combined weight of the set.

Observation 2.6.5. Let T_1 be a parse tree with an unexpanded leaf node X_{ij}, and let T_2 be a parse tree with root node of the form $\langle X_{ij} \to \lambda \rangle$. If we define

$$T = T_1 \triangle^{\cancel{X_{ij}}}_{\langle X_{ij} \to \lambda \rangle} T_2,$$

then

1. T is a valid parse tree.

2. $W_{\mathcal{G}}(T) = W_{\mathcal{G}}(T_1) W_{\mathcal{G}}(T_2)$.

2.6.3 Inner Parse Trees

Definition 2.6.6. An *inner parse tree* of C is a parse tree T in which every leaf node is a lexical node corresponding to a segment of C. Let $\mathbb{I}^C_{\mathcal{G}}(X_{ij}, \lambda)$ be the set of all inner parse trees of C with root node of the form $\langle X_{ij} \to \lambda \rangle$. Let $\mathbb{I}^C_{\mathcal{G}}(X_{ij})$ be the set of all inner parse trees of C with root node of the form $\langle X_{ij} \to \lambda \rangle$ for any $\lambda \in \mathcal{F}_{\mathcal{G}}$.

Proposition 2.6.7. *We can construct the set $\mathbb{I}^C_{\mathcal{G}}(X_{ij}, \lambda)$ recursively as follows:*

- $\mathbb{I}^C_{\mathcal{G}}(X_{ii+1}, \ell_{ii+1}) = \left\{ \frac{\langle X_{ii+1} \to \ell_{ii+1} \rangle}{\emptyset \quad | \quad \emptyset} \right\}$ *if $[X \to \ell] \in \mathcal{R}(X)$, and empty otherwise.*
- $\mathbb{I}^C_{\mathcal{G}}(X_{ij}, Y_{ik}Z_{kj}) = \left\{ \frac{\langle X_{ij} \to Y_{ik}Z_{kj} \rangle}{T_Y \quad | \quad T_Z} : T_Y \in \mathbb{I}^C_{\mathcal{G}}(Y_{ik}), T_Z \in \mathbb{I}^C_{\mathcal{G}}(Z_{kj}) \right\}.$

Proof. Let $T \in \mathbb{I}^C_{\mathcal{G}}(X_{ij})$. The root node is either of the form $\langle X_{ii+1} \to \ell_{ii+1} \rangle$ or $\langle X_{ij} \to Y_{ik}Z_{kj} \rangle$. In the former case, the root is a lexical node, and cannot have any children.

In the latter case, since T is an inner parse tree, the root node cannot be a leaf node, and it is constrained to have exactly two children, which must be of the form $\langle Y_{ik} \to \lambda_Y \rangle$ and $\langle Z_{kj} \to \lambda_Z \rangle$, since T has no unexpanded nodes. Every leaf node of either subtree is also a leaf node of T, and thus lexical. Therefore the subtrees headed by these nodes are also inner parse trees, and reside in the specified sets. $\square$

Proposition 2.6.8. *The set $\mathbb{I}^C_{\mathcal{G}}(X_{ij})$ is in one-to-one correspondence with the set of derivations of $C[i:j]$ from the placed symbol X_{ij} according to the substitution rules of $\mathcal{G}$. Moreover, for $T \in \mathbb{I}^C_{\mathcal{G}}(X_{ij})$, $W_{\mathcal{G}}(T)$ gives the probability of the derivation.*

Proof. Since inner parse trees cannot have unexpanded nodes, they correspond to complete derivations of a subcurve from the placed symbol X_{ij}. Since each step of the derivation is chosen independently from the others, the probability of the derivation represented by the parse tree is the product of the probabilities of the individual substitutions, which is given by $W_{\mathcal{G}}(T)$. $\square$

The probability that we derive a curve $C[i:j]$ from a placed symbol X_{ij} is just the sum of the probabilities of any particular derivation, taken over all possible derivations. This probability is called the *inside probability*:

$$Inside_{\mathcal{G}}^{C}(X_{ij}) = \sum_{T \in \mathbb{I}_{\mathcal{G}}^{C}(X_{ij})} W_{\mathcal{G}}(T).$$

Since $P(C \mid \mathcal{G})$ is just the total probability of deriving C from $\mathcal{G}$, it is clear by Proposition 2.6.8 that

$$P(C \mid \mathcal{G}) = Inside_{\mathcal{G}}^{C}(S_{0n}).$$

We can compute the inside probability efficiently:

Observation 2.6.9. We can compute all inside probabilities in time $O(|\mathcal{R}|n^3)$ by using the following relations:

$$\begin{aligned} Inside_{\mathcal{G}}^{C}(X_{ii+1}) &= \sum_{T \in \mathbb{I}_{\mathcal{G}}^{C}(X_{ii+1})} W_{\mathcal{G}}(T) \\ &= P_{sub}(\langle X_{ii+1} \to \ell_{ii+1} \rangle) \end{aligned}$$

by the first case of Proposition 2.6.7.

$$\begin{aligned} Inside_{\mathcal{G}}^{C}(X_{ij}) &= \sum_{T \in \mathbb{I}_{\mathcal{G}}^{C}(X_{ij})} W_{\mathcal{G}}(T) \\ &= \sum_{\substack{[X \to YZ] \in \mathcal{R}(X) \\ i<k<j}} \sum_{T \in \mathbb{I}_{\mathcal{G}}^{C}(X_{ij}, Y_{ik} Z_{kj})} W_{\mathcal{G}}(T) \end{aligned}$$

by the definition of $\mathbb{I}_{\mathcal{G}}^{C}(X_{ij})$

$$= \sum_{\substack{[X \to YZ] \in \mathcal{R}(X) \\ i<k<j}} P_{sub}(\langle X_{ij} \to Y_{ik} Z_{kj} \rangle) \cdot$$
$$\sum_{T_Y \in \mathbb{I}_{\mathcal{G}}^{C}(Y_{ik})} \sum_{T_Z \in \mathbb{I}_{\mathcal{G}}^{C}(Z_{kj})} W_{\mathcal{G}}(T_Y) W_{\mathcal{G}}(T_Z)$$

by the second case of Proposition 2.6.7

$$= \sum_{\substack{[X \to YZ] \in \mathcal{R}(X) \\ i<k<j}} P_{sub}(\langle X_{ij} \to Y_{ik} Z_{kj} \rangle) Inside_{\mathcal{G}}^{C}(Y_{ik}) Inside_{\mathcal{G}}^{C}(Z_{kj})$$

2.6.4 Outer Parse Trees

Definition 2.6.10. An *outer parse tree* of C is a parse tree T with $sym(root(T)) = S_{0n}$ in which one special leaf is an unexpanded node X_{ij}, and all other leaf nodes are lexical nodes corresponding to segments of C. Let $\mathbb{O}_{\mathcal{G}}^{C}(X_{ij})$ be the set of all outer parse trees which have unexpanded node X_{ij}.

Proposition 2.6.11. *(Illustrated in Figure 2.12) We can construct the set $\mathbb{O}_{\mathcal{G}}^{C}(X_{ij})$ recursively as follows:*

- $\mathbb{O}_{\mathcal{G}}^{C}(X_{0n}) = \left\{ \begin{array}{c} S_{0n} \\ \hline \emptyset \quad | \quad \emptyset \end{array} \right\}$, *if $X = S$, and is empty otherwise.*

•

$$\mathbb{O}^C_{\mathcal{G}}(X_{ij}) =$$

$$\bigcup_{\substack{h<i \\ [Z\to YX]\in\mathcal{R}}} \left\{ T_{out} \triangle^{\cancel{Z_{hj}}}_{\langle Z_{hj}\to Y_{hi}X_{ij}\rangle)} \frac{\langle Z_{hj} \to Y_{hi}X_{ij}\rangle}{T_Y \qquad | \qquad X_{ij}} : \right.$$

$$\left. T_{out} \in \mathbb{O}^C_{\mathcal{G}}(Z_{hj}), T_Y \in \mathbb{I}^C_{\mathcal{G}}(Y_{hi}) \right\} \bigcup$$

$$\bigcup_{\substack{i<k \\ [Z\to XY]\in\mathcal{R}}} \left\{ T_{out} \triangle^{\cancel{Z_{ik}}}_{\langle Z_{ik}\to X_{ij}Y_{jk}\rangle} \frac{\langle Z_{ik} \to X_{ij}Y_{jk}\rangle}{X_{ij} \qquad | \qquad T_Y} : \right.$$

$$\left. T_{out} \in \mathbb{O}^C_{\mathcal{G}}(Z_{ik}), T_Y \in \mathbb{I}^C_{\mathcal{G}}(Y_{jk}) \right\}$$

(For notational simplicity above, we are writing X_{ij} in place of $\frac{X_{ij}}{\emptyset \quad | \quad \emptyset}$.)

Proof. Let $T \in \mathbb{O}^C_{\mathcal{G}}(X_{ij})$. The unexpanded node X_{ij} may be the root of T, in which case X_{ij} must be S_{0n}, and we are in the first case.

Otherwise, X_{ij} is not the root of T, and has a parent which is either $\langle Z_{hj} \to Y_{hi}X_{ij}\rangle$, or $\langle Z_{ik} \to X_{ij}Y_{jk}\rangle$. Let us restrict to the second case; the first case is strictly analogous.

If we remove the subtree rooted at Y_{jk} and replace $\langle Z_{ik} \to X_{ij}Y_{jk}\rangle$ with an unexpanded node Z_{ik}, we get a tree T_{out} which is a valid outer tree for Z_{ik}:

- T_{out} has a single unexpanded node Z_{ik}
- $root(T_{out}) = root(T) = S_{0n}$.

Furthermore, the subtree T_Y rooted at Y_{jk} satisfies $sym(root(T_Y)) = Y_{jk}$, and is an inner parse tree, since all remaining leaf nodes are lexical nodes. This completes the proof. □

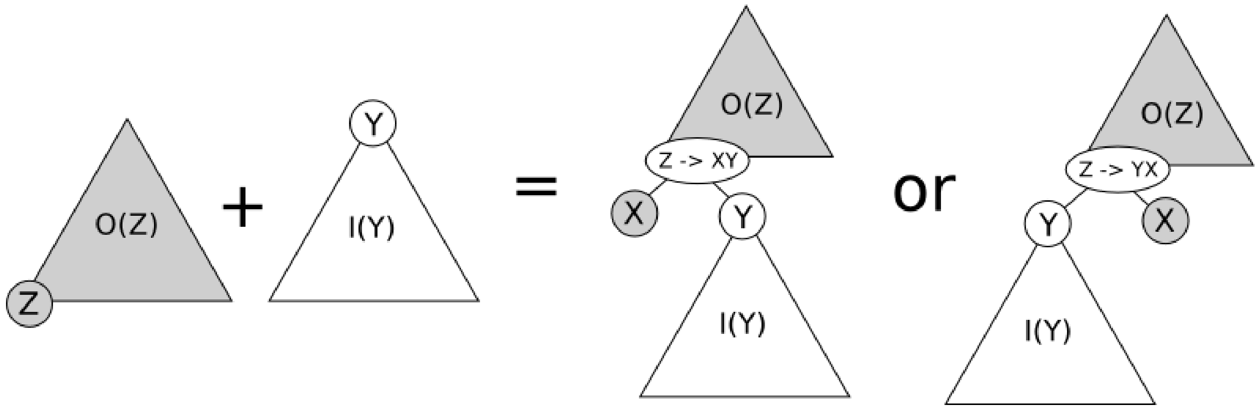


Figure 2.12: The two ways of forming an outer parse tree for X. Here shaded ovals are unexpanded nodes, shaded triangles are outer parse trees, and unshaded triangles are inner parse trees.

We define the *outside weight* as:

$$Outside_{\mathcal{G}}^{C}(X_{ij}) = \sum_{T \in \mathbb{O}_{\mathcal{G}}^{C}(X_{ij})} W_{\mathcal{G}}(T).$$

This gives the total weight of parse trees corresponding to derivations of the curvilinear form $\ell_{01} \dots \ell_{i-1i} X_{ij} \ell_{jj+1} \dots \ell_{n-1n}$ from S_{0n}.

Observation 2.6.12. Given the inside probabilities, we can compute the outside weights in time $O(|\mathcal{R}|n^3)$ by using the following relations (note that we can do this with dynamic programming because we can compute the values in a fixed order, based on the length of the intervals):

$$\begin{aligned} Outside_{\mathcal{G}}^{C}(S_{0n}) &= \sum_{T \in \mathbb{O}_{\mathcal{G}}^{C}(S_{0n})} W_{\mathcal{G}}(T) \\ &= W_{\mathcal{G}}\left(\frac{S_{0n}}{\emptyset \quad | \quad \emptyset} \right) = 1 \end{aligned}$$

by the first case of Proposition 2.6.11.

$$
\begin{aligned}
Outside_{\mathcal{G}}^{C}(X_{ij}) &= \sum_{T \in \mathbb{O}_{\mathcal{G}}^{C}(X_{ij})} W_{\mathcal{G}}(T) \\
&= \sum_{\substack{h<i \\ [Z \to YX] \in \mathcal{R}}} P_{sub}(\langle Z_{hj} \to Y_{hi} X_{ij} \rangle) \cdot \\
&\qquad \sum_{T_{out} \in \mathbb{O}_{\mathcal{G}}^{C}(Z_{hj})} \sum_{T_Y \in \mathbb{I}_{\mathcal{G}}^{C}(Y_{hi})} W_{\mathcal{G}}(T_{out}) W_{\mathcal{G}}(T_Y) \\
&+ \sum_{\substack{j<k \\ [Z \to XY] \in \mathcal{R}}} P_{sub}(\langle Z_{ik} \to X_{ij} Y_{jk} \rangle) \cdot \\
&\qquad \sum_{T_{out} \in \mathbb{O}_{\mathcal{G}}^{C}(Z_{ik})} \sum_{T_Y \in \mathbb{I}_{\mathcal{G}}^{C}(Y_{jk})} W_{\mathcal{G}}(T_{out}) W_{\mathcal{G}}(T_Y)
\end{aligned}
$$

by the second case of Proposition 2.6.11.

$$
\begin{aligned}
&= \sum_{\substack{h<i \\ [Z \to YX] \in \mathcal{R}}} P_{sub}(\langle Z_{hj} \to Y_{hi} X_{ij} \rangle) Outside_{\mathcal{G}}^{C}(Z_{hj}) Inside_{\mathcal{G}}^{C}(Y_{hi}) \\
&+ \sum_{\substack{j<k \\ [Z \to XY] \in \mathcal{R}}} P_{sub}(\langle Z_{ik} \to X_{ij} Y_{jk} \rangle) Outside_{\mathcal{G}}^{C}(Z_{ik}) Inside_{\mathcal{G}}^{C}(Y_{jk})
\end{aligned}
$$

2.6.5 Dealing with Closed Curves

We can think of a closed curve as an open curve whose endpoints are the same. However, we will prefer to think of it in a different way. We wish to preserve the symmetry of closed curves by considering any rotation of the curve to be equivalent. We can then define an "opening" operation, which takes a closed curve $C[0], \ldots, C[n-1]$, and an integer k, and produces the open curve $C[k], C[k+1], \ldots, C[n-1], C[0], \ldots, C[k]$ whose endpoints are the same.

When given a closed curve, we will assume that any of the n possible opening operations are equally likely, and thus have probability $\frac{1}{n}$. Alternatively, we can imagine that the closed curve has had one of the n possible rotations applied to it before we see it, with each rotation being equally likely.

Given a closed curve, we could then apply the preceding machinery to every possible opening of the curve. However, this is grossly inefficient, since different openings of the curve still share many of the same subcurves. Here, we describe a way to take maximum advantage of this sharing.

Remark 2.6.13. For closed curve grammars, the top-level midpoint distributions $\mu_{S \to XY}$ (defined in Section 2.4) cannot be proper distributions if our grammar is going to be invariant under similarity transformations (translation, scaling, and rotation).

The symbol S will be realized as a placed symbol $S_{p,p}$, and the placed rule will be of the form $S_{p,p} \to X_{p,q}Y_{q,p}$. Then, any choice of midpoint q is equally good, since we can map the pair (p, q) to a pair (p, q') via a similarity transformation. This is a problem because we cannot define a uniform distribution on all of $\mathbb{R}^2$!

We handle this by setting $\mu_{S \to XY}(\cdot) = 1$ in our formulas. We justify this by thinking of each curve as being a representative of its equivalence class under similarity transformations.

The probability that we see a parse tree is therefore $P(C, T \mid \mathcal{G}) = \frac{1}{n} W_{\mathcal{G}}(T)$. In order to describe the parse trees, we must allow placed symbols X_{ij}, where $j < i$. In this case, X_{ij} will be understood to be a placed symbol that ultimately expands to be all the indices between 0 and n that are after i or before j. To simplify matters, we introduce clockwise intervals:

$$cw_n(i,j) = \begin{cases} \{k \in \mathbb{Z} \mid i < k < j\} & i < j \\ \{k \in \mathbb{Z} \mid 0 \le k < j \text{ or } i < k < n\} & i > j \\ \{k \in \mathbb{Z} \mid 0 \le k < n, k \neq i\} & i = j \end{cases} .$$

These are the indices in between i and j on the curve, where we move "clockwise" in the

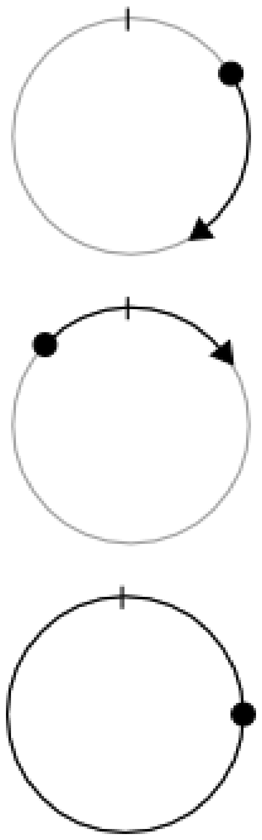


Figure 2.13: The three kinds of clockwise intervals.

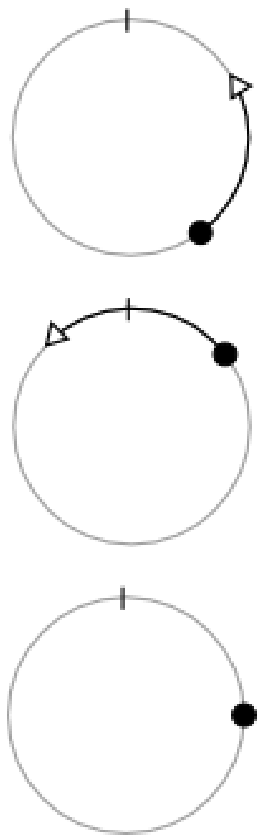


Figure 2.14: The three kinds of counter-clockwise intervals.

direction of increasing indices. We also introduce counter-clockwise intervals

$$ccw_n(i,j) = \begin{cases} \{k \in \mathbb{Z} \mid 0 \leq k < i \text{ or } j < k < n\} & i < j \\ \{k \in \mathbb{Z} \mid j < k < i\} & i > j \\ \emptyset & i = j \end{cases} .$$

These are the points which are outside of the clockwise interval between i and j, excluding i and j.

We can then adapt Observations 2.6.9 and 2.6.12 for closed curves as follows:

Proposition 2.6.14. *We can recursively compute inside weights for closed curves in time $O(|\mathcal{R}|n^3)$:*

$$
\begin{aligned}
CInside_{\mathcal{G}}^{C}(X_{ii+1}) &= P_{sub}(\langle X_{ii+1} \to \ell_{ii+1}\rangle) \\
CInside_{\mathcal{G}}^{C}(X_{(n-1)0}) &= P_{sub}(\langle X_{(n-1)0} \to \ell_{(n-1)0}\rangle) \\
CInside_{\mathcal{G}}^{C}(X_{ij}) &= \sum_{\substack{[X\to YZ]\in\mathcal{R}(X)\\ k\in cw_n(i,j)}} P_{sub}(\langle X_{ij} \to Y_{ik}Z_{kj}\rangle) CInside_{\mathcal{G}}^{C}(Y_{ik}) CInside_{\mathcal{G}}^{C}(Z_{kj}).
\end{aligned}
$$

Proposition 2.6.15. *We can recursively compute outside weights for closed curves in time $O(|\mathcal{R}|n^3)$:*

$$
\begin{aligned}
COutside_{\mathcal{G}}^{C}(S_{ii}) &= \frac{1}{n} \\
COutside_{\mathcal{G}}^{C}(X_{ij}) &= \sum_{\substack{h\in ccw_n(i,j)\\ [Z\to YX]\in\mathcal{R}}} P_{sub}(\langle Z_{hj} \to Y_{hi}X_{ij}\rangle) COutside_{\mathcal{G}}^{C}(Z_{hj}) CInside_{\mathcal{G}}^{C}(Y_{hi}) \\
&+ \sum_{\substack{k\in ccw_n(i,j)\\ [Z\to XY] in \mathcal{R}}} P_{sub}(\langle Z_{ik} \to X_{ij}Y_{jk}\rangle) COutside_{\mathcal{G}}^{C}(Z_{ik}) CInside_{\mathcal{G}}^{C}(Y_{jk}).
\end{aligned}
$$

2.7 Building a Grammar From a Single Curve

The current work is based on Felzenszwalb and Schwartz [2007], in which grammar-like structures were constructed from single training curves. We follow that approach here.

A grammar $\mathcal{G}_C$ for a curve C should produce C, and curves which look like C. We are not trying to achieve structural variation yet, so the only variation that $\mathcal{G}_C$ needs to account for is:

- Slight geometric deformations of C
- Curves like C that have more or fewer sample points.

Geometric deformation is modeled with a midpoint distribution that is defined in terms of C. We discuss our models in Section 2.8. Curves with fewer sample points are modeled by including, for every nonterminal $X \in \mathcal{N}$, the rule $[X \to \ell]$. We initially set the probability of this rule based on the scale of X, as described in Section 2.9.

The nonterminals of our grammar will correspond to subcurves of C. We model curves with more sample points than C by allowing any nonterminal X that arises from a subcurve of length 1 to have two rules:

- $X \to \ell$
- $X \to XX$

The second rule is infinitely recursive, and thus allows the curve to get arbitrarily long.

2.7.1 General Case

Let C be a curve of length n. A *generic parse tree* for C is a parse tree that does not refer to any particular grammar. This will just be a binary tree T where each node is labeled by an interval (i, j), $0 \leq i < j \leq n$, and

- The root of T is $(0, n)$
- If (i, j) is a node, and $j - i \geq 2$, then (i, j) has exactly two children, and they are of the form (i, k) and (k, j) for some k.
- T has n leaf nodes $(i, i + 1)$ for $i = 0, \ldots, n - 1$.

We can then specify a simple grammar for C that produces the parse tree T. For every node (i, j) in T, $\mathcal{G}$ has a nonterminal $X^{(i,j)}$. When (i, j) has children (i, k) and (k, j), $\mathcal{G}$ has a rule $[X^{(i,j)} \to X^{(i,k)} X^{(k,j)}]$. The scale of the nonterminal $X^{(i,j)}$ is $\frac{j-i}{n}$ (see Section 2.9).

Our choice of an initial grammar for C is thus based on choosing a generic parse for C. Some choices:

1. An arbitrary parse, chosen to make the tree as balanced as possible. This approach corresponds most closely to that of Felzenszwalb and Schwartz [2007].

2. A parse that has been chosen to respect the natural part structure of C. This approach would require us to identify natural constituents of curves. There has been some work on this question, e.g. Richards et al. [1985].

3. We can choose multiple parses, and create a grammar that can parse C in multiple ways.

Our favored approach is number 3. We want to create a small grammar that can parse C in as many ways as possible, so that the EM retraining can discover the "correct" sub-grammar.

2.8 Models of Triangle Deformation

When sampling from our grammar, we place a midpoint q according to the midpoint distribution $\mu_{X \to YZ}(q; p, r)$, where $X_{p,r}$ is the placed symbol we are currently replacing, and $X \to YZ$ is the substitution rule we are using.

When building a grammatical model from a single curve C, we base the distribution $\mu_{X^{(i,k)} \to X^{(i,j)} X^{(j,k)}}$ on the triangle $C[i], C[j], C[k]$, with $C[i], C[j], C[k]$ playing the roles of p, q, and r respectively.

We have experimented with two different approaches to modeling the shape of triangles, the Watson distribution and a non-parametric distribution. Unless otherwise specified, midpoint distributions will always be modeled with the Watson distribution.

2.8.1 Procrustes Distance and The Watson Distribution

One useful midpoint distribution is given by the complex Watson distribution, which is a probabilistic analogue of the Procrustes distance.

Let us represent our points (x, y) in $\mathbb{R}^2$ as complex numbers $x + iy$, and ordered triples of points by vectors of length three. For every such vector $z = \begin{pmatrix} p & q & r \end{pmatrix}$, we define a canonical version

$$\widehat{z} = \frac{z - w}{\|z - w\|},$$

where $w = \begin{pmatrix} \frac{p+q+r}{3} & \frac{p+q+r}{3} & \frac{p+q+r}{3} \end{pmatrix}$. We do this transformation to achieve invariance to scale and location, since, if $y = az + b$ for $a \in \mathbb{R}$, then $\widehat{y} = \widehat{z}$.

The (complex) Watson distribution is then defined as follows:

$$Watson(z; \mu, \kappa) = \frac{1}{Z(\kappa)} e^{\kappa |\widehat{z}^* \widehat{\mu}|^2},$$

where μ is the mode of the distribution, and $\kappa > 0$ is a concentration parameter. This distribution is invariant to rotation since, if $y = e^{i\theta} z$, $|\widehat{y}^* \widehat{\mu}| = |\widehat{z}^* \widehat{\mu}|$. More details on the Watson distribution can be found in Dryden and Mardia [1998].

The Watson distribution can be thought of as a probabilistic version of the Procrustes distance, which can be defined as

$$d_P(y, z) = \sqrt{1 - \frac{z^* y y^* z}{y^* y z^* z}},$$

and thus

$$Watson(z; \mu, \kappa) \propto e^{-\kappa \cdot d_P(\widehat{z}, \widehat{\mu})^2},$$

where we are using the fact that $\widehat{z}^* \widehat{z} = \|\widehat{z}\|^2 = 1$.

2.8.2 Non-Parametric Deformation Model

We have also experimented with the nonparametric distribution given by the Parzen windows method (Parzen [1962]). We want our grammar to be invariant to simultaneous translation, scale, and rotation of the points p, q, r. Therefore, we translate, scale, and rotate $\mathbb{R}^2$ with a map ϕ such that $\phi(p) = (0, 0)$ and $\phi(r) = (1, 0)$, and represent q via the coordinates $\widehat{q} = \phi(q)$.

If we have seen samples $q_1, \ldots, q_k$, then

$$\mu_{X \to YZ}(q; p, r) = \frac{1}{n} \sum_{i=1}^{k} \frac{1}{2\pi h^2} e^{\frac{\|\widehat{q_i} - \widehat{q}\|^2}{2h^2}}.$$

In the context of kernel density estimators, the parameter h is called the bandwidth. It specifies how much smoothing to apply to the density estimation. Selecting a suitable bandwidth is often problematic.

It is worth noting that this nonparametric distribution is just a mixture of Gaussians, and thus there is no solid distinction between a mixture of multiple copies of the rule $X \to YZ$ with Gaussian midpoint distributions, and a single copy with a mixture of Gaussians midpoint distribution. We will prefer the second, since the resulting grammar has a simpler structure, as discussed in Chapter 5.

2.9 Dealing with Variation in Length

As mentioned before, our grammar models longer curves with rules of the form $X \to XX$, and shorter curves with rules of the form $X \to \ell$. When building our initial models, it is very important to assign reasonable probabilities to these rules, because we want to use these rules in parsing, but prevent pathological parses. For example, we don't want to parse an input curve in such a way that many high-level nonterminals X use the rule $X \to \ell$ and some low-level nonterminal Y uses the rule $Y \to YY$ many times. Since the balance between different rules is governed by a multinomial distribution, we need a reasonable prior on multinomial distributions that prevents pathological parses.

Let X be a nonterminal. When there is no prior reason to believe that any $\rho_X(X \to \lambda)$ is larger than the others, it is natural to use a Dirichlet prior with each α_i the same. For shape grammars, this is not entirely the case; since rules of the form $[X \to YZ]$ and $[X \to \ell]$ serve different roles in the grammar, we expect their probabilities to take on different values.

In particular, if the nonterminal X is meant to model long curves, it is unlikely that a long curve will have only two sample points, especially if we are parsing a curve that has many sample points. Therefore, we would expect the probability of $[X \to \ell]$ to be low.

Let C be a curve. We define the *scale* of a subcurve C' to be $s(C') = |C'|/|C|$. We can then specify an ideal scale s_X for the nonterminal X. Let $k = |\mathcal{R}(X)|$. Our prior distribution on ρ_X is then $Dir(\alpha_1, \ldots, \alpha_k)$, where

$$\begin{aligned} \alpha_1 &= k \cdot \alpha_{dir} \cdot e^{-\omega s_X^2} \\ \alpha_i &= k \cdot \alpha_{dir} \cdot \frac{1 - e^{-\omega s_X^2}}{k-1} && 2 \le i \le k. \end{aligned}$$

Here α_1 corresponds to the rule $[X \to \ell]$. Suitable values of α_{dir} are probably significantly less than one, in light of Johnson et al. [2007]. A suitable value for ω is more complicated, and depends on the value of other model parameters.

This prior is suitable, because it produces grammars biased towards balanced parses. To see this, consider a grammar $\mathcal{G}$ whose parameters are set according to this prior. (This will be true of our initial grammar, as we describe in Section 2.7.) Let C be a curve of length n, and T be a $\mathcal{G}$-parse of C. T will then specify nonterminals $X_1, \ldots, X_n$ to parse each line segment of C. Thus, the likelihood $P(C, T \mid \mathcal{G})$ will contain factors $\rho_{X_i}([X_i \to \ell])$ for $i = 1, \ldots, n$. If $\rho_{X_i}([X_i \to \ell])$ is $e^{-\omega s_X^2}$, then these factors contribute

$$e^{-\omega \sum_i s_{X_i}^2}$$

in total. Since the X_i combine to parse C, the scales s_{X_i} will hopefully sum to approximately one. In this case the probability will be highest when $\sum_i s_{X_i}^2$ is lowest, which occurs when all the s_{X_i} are equal. The stated Dirichlet prior thus pushes $\mathcal{G}$ to favor balanced parse trees, as desired.

2.10 Parsing to Recover Correspondences

Given a grammar model and a curve, we want to calculate how likely the curve is under the model, and we would like to extract the most likely correspondence between the points of the curve and the parts of the model. We call this process parsing, because it is extremely similar to CKY parsing with standard PCFG's. We defer the technical details of parsing until a later chapter.

We examine parsing in this section by looking at human silhouettes from a dataset generated by Romer Rosales. We have two versions of this dataset: one where a silhouette has been automatically extracted by subtracting the background, and the curve simplified slightly by the discretization method discussed in Section 2.3; and a hand-annotated version, where we have marked by hand the position of 27 corresponding points on the body in each image.

Throughout this section, we examine the task of building a grammatical model from one curve, and using it to parse another curve. We demonstrate the accuracy of these parses by showing the correspondence between the points of the two curves induced by the most likely parse.

The easiest parsing task is to recover a one-to-one correspondence between two curves that have the same number of points, which we demonstrate in Figure 2.15. Because there are no missing or extra points, this is straightforward. The two curves are both from the hand-annotated Romer dataset. The grammar is built using a hand-picked set of constituents.

A somewhat harder task is to parse a curve that is longer than the curve on which our model is based. We wish to recover a reasonable correspondence between the points of the coarse model curve and a subset of the points of the finer input curve. We do this by building a grammar model from the coarse curve and using it to parse the finer curve. The grammar must have lengthening rules, but doesn't need shortening rules. This demonstrates that we can model longer curves than were used to build the grammar. This is shown in Figure 2.16, where a model built from a hand-annotated curve is used to parse a curve from the

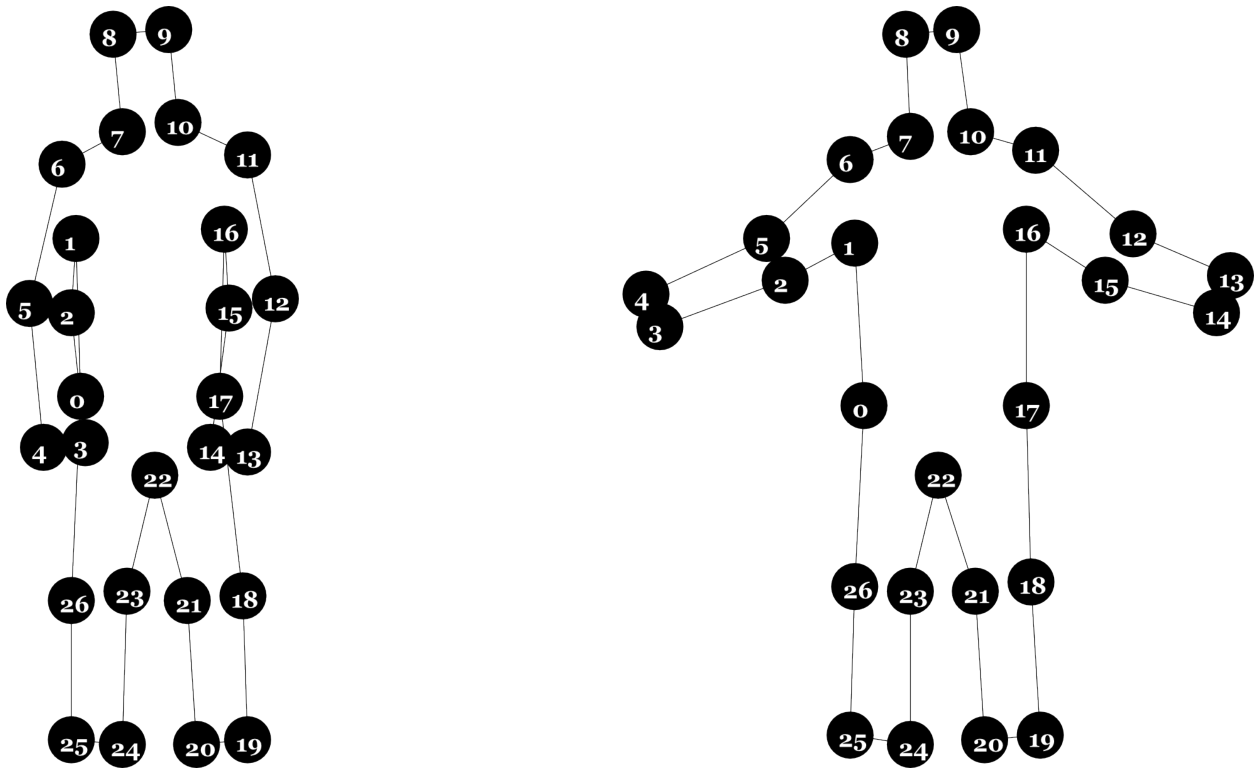


Figure 2.15: Recovering a One-to-one Correspondence

ground-truth Romer dataset. We successfully recover a very reasonable correspondence.

We also wish to recover a reasonable correspondence by building a grammar from a longer (i.e., finer) model curve and parsing a shorter (i.e., coarser) input curve. This demonstrates that we can model shorter curves than were used to build the grammar. This task is generally harder than the prior task.

Here we build a grammar from a ground-truth Romer curve, and try to parse one of the (much shorter) hand-annotated Romer curves. We can safely assume that every point in the parsed curve has a corresponding one in the example curve, which is the reverse of the previous experiments. In order to do this successfully, the grammar needs shortening rules, but not lengthening rules.

In Figure 2.17, we show the results of parsing a shorter curve with very little geometric variation. The fine details are not quite right, but overall the correspondence is reasonable. In Figure 2.18, we show the results of parsing a shorter curve with significant geometric

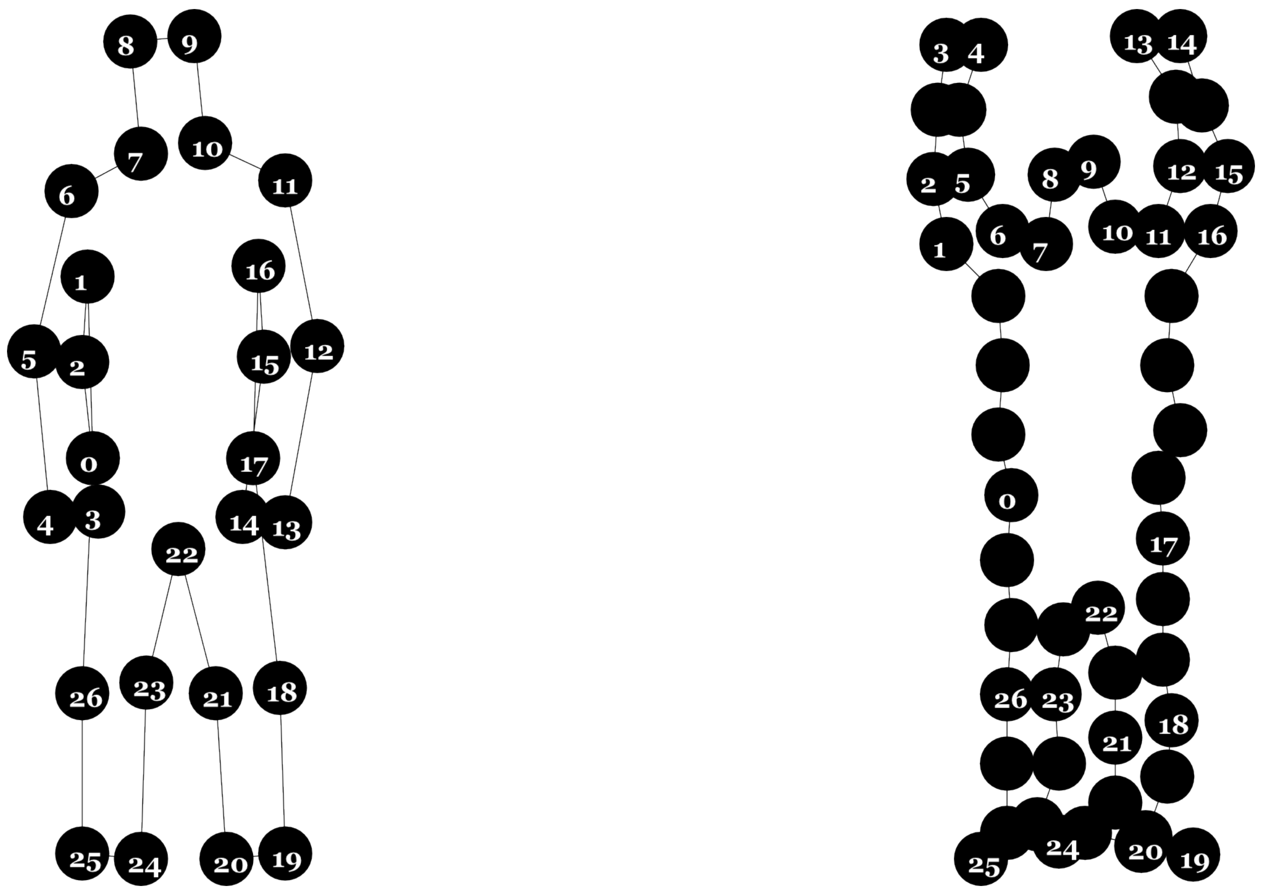


Figure 2.16: Parsing a Longer Curve

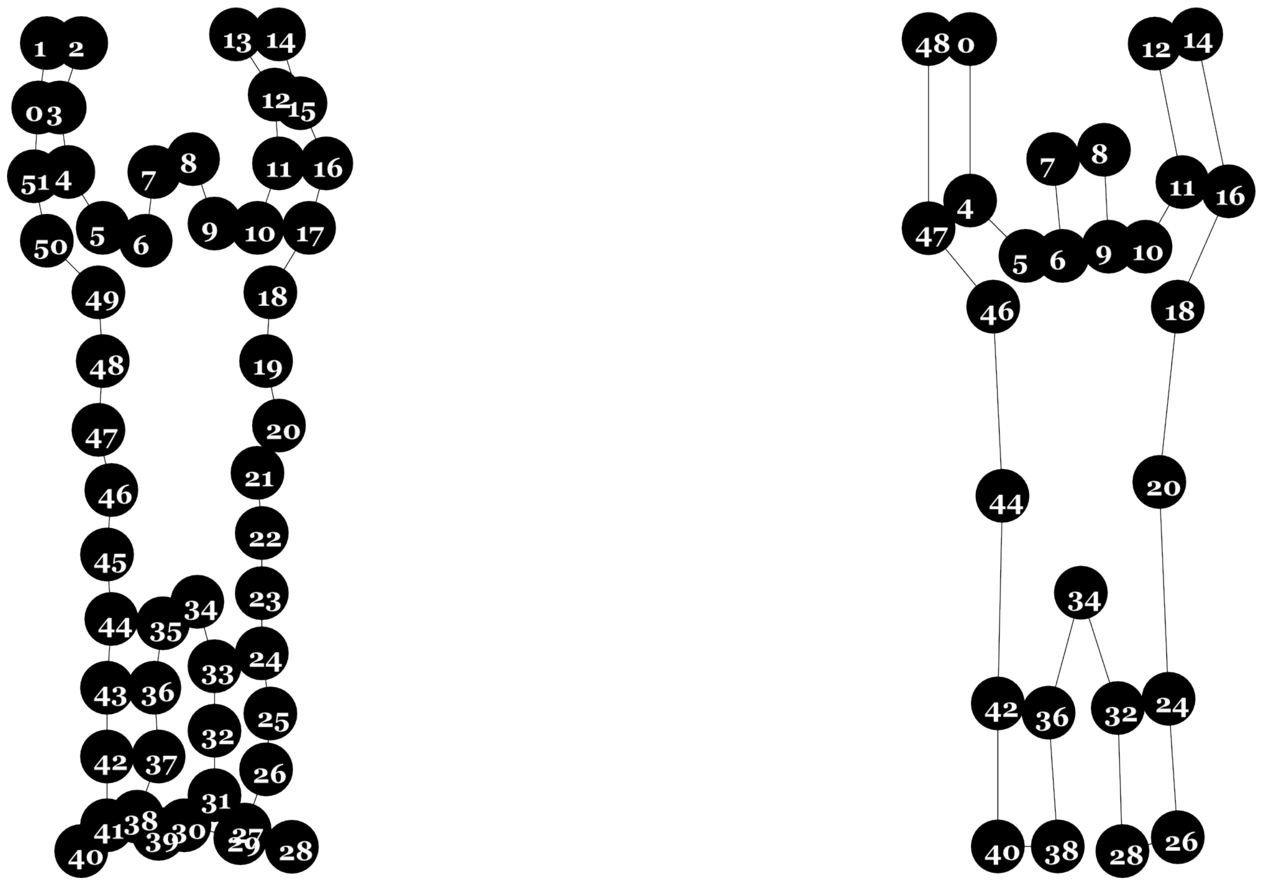
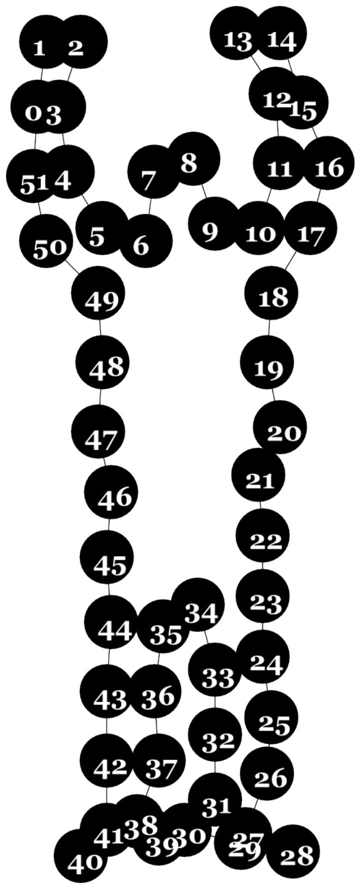


Figure 2.17: Parsing a shorter curve with very little geometric variation.

variation. In this case, the correspondence found is completely incorrect.

Our models are very sensitive to the initial decomposition specified for the model curve. The two experiments above used a balanced but otherwise arbitrary decomposition of the model curve. By using a more carefully chosen decomposition, we get much better parses, which can be seen in Figures 2.19 and 2.20. In Figure 2.19, the fine details are now exactly what we would expect, in contrast to Figure 2.17. In Figure 2.20, even though we have the same significant geometric variation as in Figure 2.18, we now find a very good correspondence.



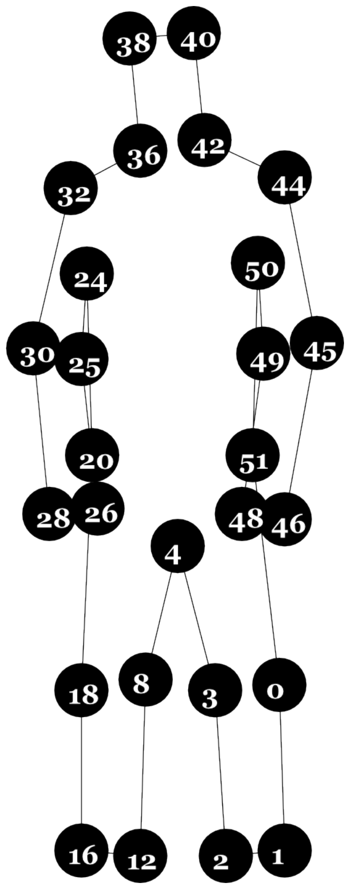


Figure 2.18: Parsing a shorter curve with significant geometric variation.

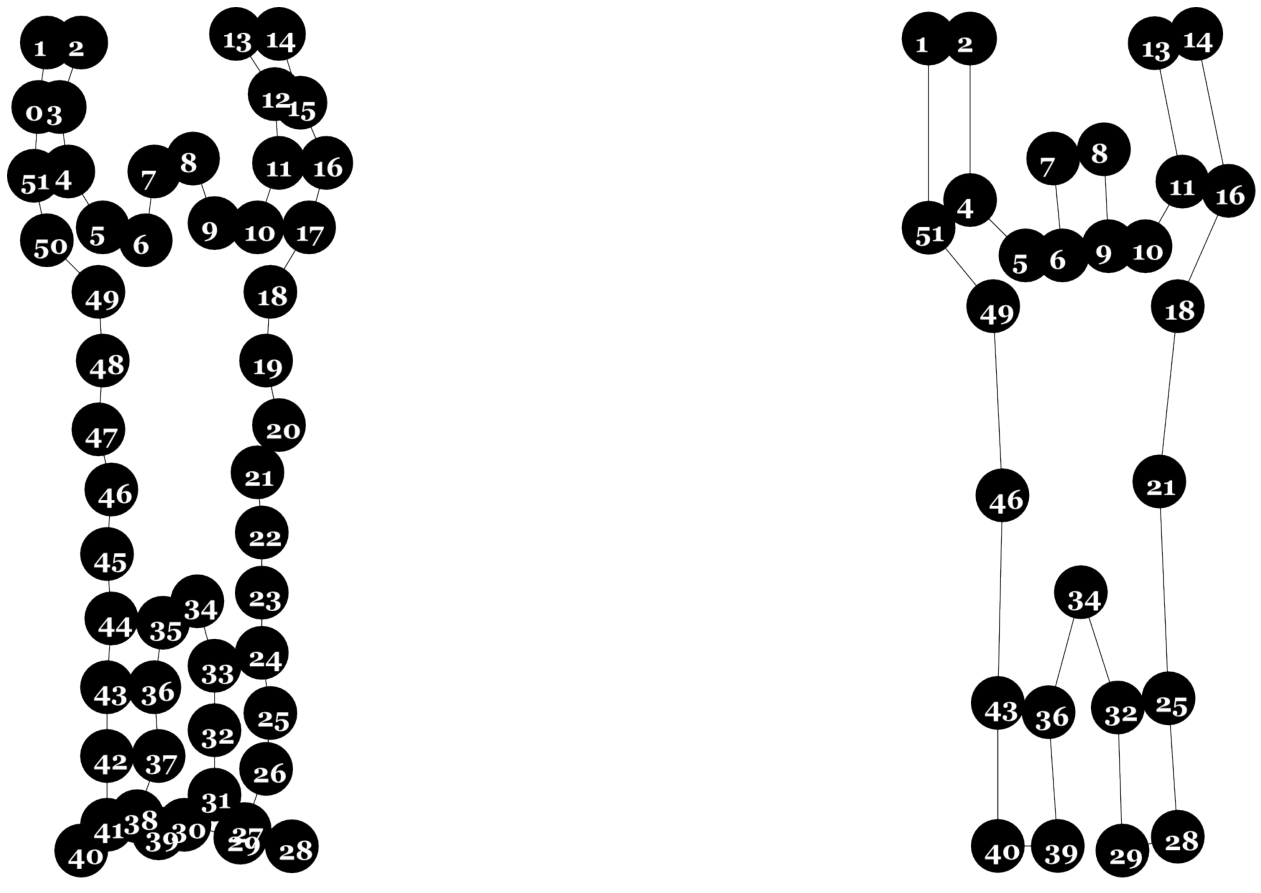


Figure 2.19: Parsing a shorter curve with very little geometric variation.

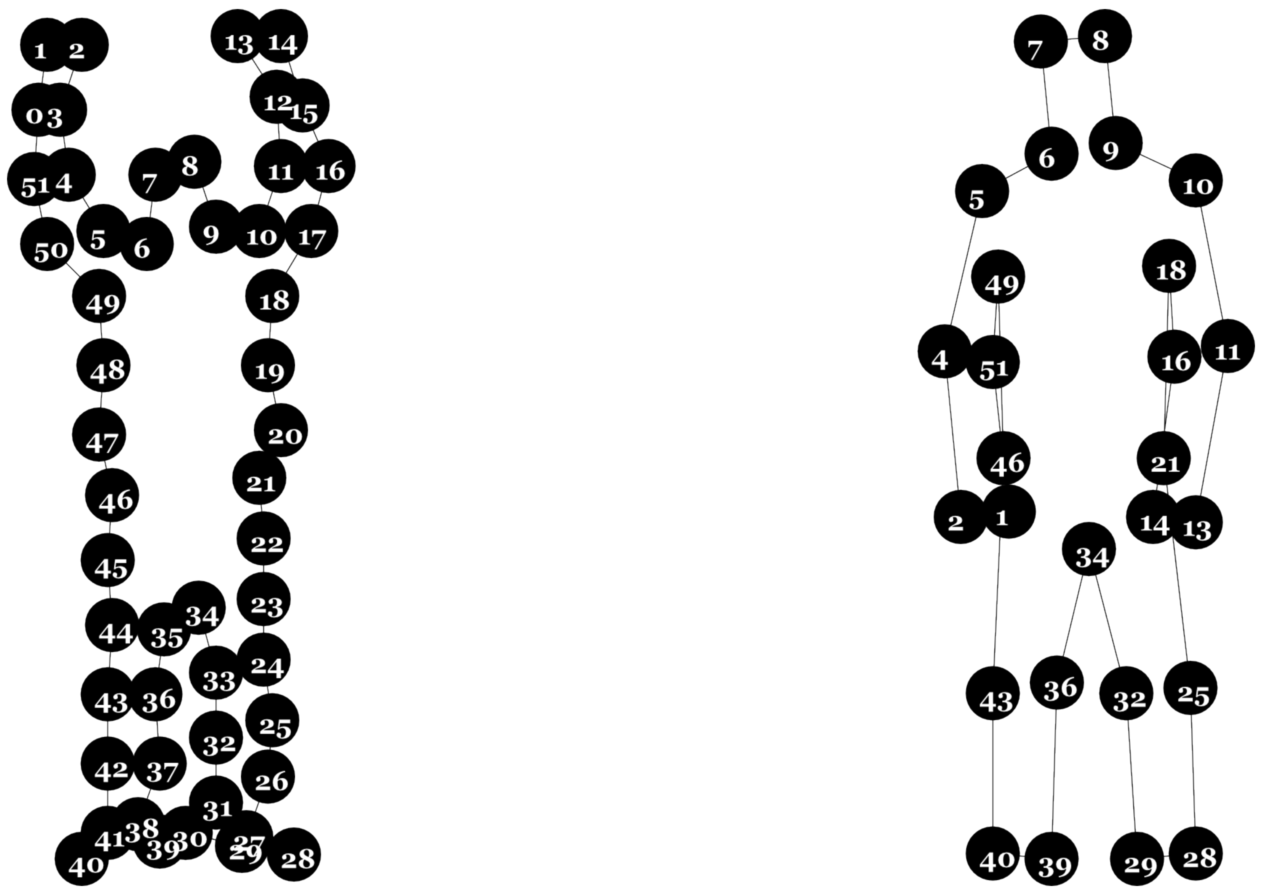


Figure 2.20: Parsing a shorter curve with significant geometric variation.

Thus, even on fairly straightforward parsing tasks, our models are extremely sensitive to the particular hierarchical structure that we impose on the curve. The shortening rules only allow the parser to chop off constituents. Therefore, it is important to have good constituents. We address this in Chapter 3 by using multiple decompositions, and learn which constituents allow us to best explain our data.

CHAPTER 3
PARAMETER LEARNING FOR GRAMMATICAL MODELS

In this chapter, we describe a mathematically rigorous way to tune our model parameters using the EM algorithm. We give the details of the algorithm in Section 3.1. In the remainder of the chapter, we show samples from various grammatical models after training, in order to demonstrate that we have learned a correct and interesting model. Finally, in Section 3.8, we show that samples from standard models fail to model the dataset we are using.

3.1 Learning Grammar Parameters with the EM Algorithm

Let $C_1, \ldots, C_n$ be independent samples from a grammar $\mathcal{G}$, for which we know the structure, but not the parameters. We would like to set the parameters of $\mathcal{G}$ such that the posterior $P(\mathcal{G} \mid C_1, \ldots, C_n)$ is maximized. We can write the posterior as

$$\begin{aligned} P(\mathcal{G} \mid C_1, \ldots, C_n) &= P(\mathcal{G}) \cdot \frac{\prod_i P(C_i \mid \mathcal{G})}{\prod_i P(C_i)} \\ &\propto P(\mathcal{G}) \cdot \prod_i \sum_{T \in Parse_{\mathcal{G}}(C_i)} P(C_i, T \mid \mathcal{G}) \\ &= P(\mathcal{G}) \cdot \sum_{\{T_i\}} \prod_i P(C_i, T_i \mid \mathcal{G}), \end{aligned}$$

where this last sum is taken over all collections of one parse tree for each curve.

Unfortunately, it is not known how to maximize this posterior, or even the likelihood. The likelihood is known to have many local maxima in the case of context-free grammars for natural languages (Charniak [1996]). Therefore, we are forced to use approximation techniques. We use the Expectation-Maximization algorithm (Dempster et al. [1977]), which is a standard approach to finding maximum likelihood or maximum a posteriori estimates in the case where important variables are unobserved. In our case, the unobserved variables are the parse trees $\{T_i\}$.

Let $\mathbf{C} = (C_1, \ldots, C_n)$, and let $\mathbf{T}$ be the latent variable $(T_1, \ldots, T_n)$. We then iteratively find better and better grammars $\mathcal{G}^{(t)}$:

1. **E step:** Let

$$\begin{aligned} Q^{(t)}(\mathcal{G}) &= \sum_{\mathbf{T}} \left[P(\mathbf{T} \mid \mathbf{C}, \mathcal{G}^{(t)}) \log P(\mathbf{C}, \mathbf{T} \mid \mathcal{G}) \right] + \log P(\mathcal{G}) \\ &= E_{\mathbf{T} \sim \mathbf{T} \mid \mathbf{C}, \mathcal{G}^{(t)}} \left[\log P(\mathbf{C}, \mathbf{T} \mid \mathcal{G}) \right] + \log P(\mathcal{G}) \end{aligned}$$

2. **M step:** Let

$$\mathcal{G}^{(t+1)} = \operatorname*{argmax}_{\mathcal{G}} Q^{(t)}(\mathcal{G})$$

3.1.1 The E Step

$$Q^{(t)}(\mathcal{G}) = E_{\mathbf{T} \sim \mathbf{T}, \mathcal{G}^{(t)}} [\log P(C, T \mid \mathcal{G})] + \log P(\mathcal{G})$$

Since $P(\mathbf{C}, \mathbf{T} \mid \mathcal{G}) = \prod_a P(C_a, T_a \mid \mathcal{G})$, and by linearity of expectation:

$$Q^{(t)}(\mathcal{G}) = \sum_a E_{T_a \sim \Delta_a^{(t)}} [\log P(C_a, T_a \mid \mathcal{G})] + \log P(\mathcal{G}),$$

where $\Delta_a^{(t)}(T_a)$ is the distribution $P(T_a \mid C_a, \mathcal{G}^{(t)})$.

Let $Q_a^{(t)}(\mathcal{G}) = E_{T_a \sim \Delta_a^{(t)}} [\log P(C_a, T_a \mid \mathcal{G})]$. We can decompose $P(C_a, T_a \mid \mathcal{G})$ by using Proposition 2.6.8 [1]:

1. If $\mathcal{G}$ is a grammar for closed curves, $P(C_a, T_a \mid \mathcal{G})$ will have an extra factor of $\frac{1}{|C|}$. We can ignore this because the logarithm turns it into an additive constant, which is irrelevant in the M step.

$$
\begin{aligned}
Q_a^{(t)}(\mathcal{G}) &= E_{T_a \sim \Delta_a^{(t)}} \left[\log \prod_{\langle X_{ij} \to \lambda \rangle \in T_a} P_{sub}(\langle X_{ij} \to \lambda \rangle) \right] \\
&= E_{T_a \sim \Delta_a^{(t)}} \left[\sum_{\langle X_{ij} \to \lambda \rangle \in T_a} \log P_{sub}(\langle X_{ij} \to \lambda \rangle) \right]
\end{aligned}
$$

We can rewrite this as a sum over all possible concrete substitution rules, using indicator variables to eliminate absent terms:

$$
\begin{aligned}
&= E_{T_a \sim \Delta_a^{(t)}} \Bigg[\sum_{i<j<k} \sum_{[X \to YZ] \in \mathcal{R}} \mathbf{1}[\langle X_{ik} \to Y_{ij} Z_{jk} \rangle] \quad \log P_{sub}(\langle X_{ik} \to Y_{ij} Z_{jk} \rangle) \\
&\qquad + \quad \sum_{i} \sum_{[X \to \ell] \in \mathcal{R}} \mathbf{1}[\langle X_{ii+1} \to \ell_{ii+1} \rangle] \quad \log P_{sub}(\langle X_{ii+1} \to \ell_{ii+1} \rangle) \Bigg] \\
&= \sum_{i<j<k} \sum_{[X \to YZ] \in \mathcal{R}} \log P_{sub}(\langle X_{ik} \to Y_{ij} Z_{jk} \rangle) \quad E_{T_a \sim \Delta_a^{(t)}}[\mathbf{1}[\langle X_{ik} \to Y_{ij} Z_{jk} \rangle]] \\
&+ \sum_{i} \sum_{[X \to \ell] \in \mathcal{R}} \log P_{sub}(\langle X_{ii+1} \to \ell_{ii+1} \rangle) \quad E_{T_a \sim \Delta_a^{(t)}}[\mathbf{1}[\langle X_{ii+1} \to \ell_{ii+1} \rangle]]
\end{aligned}
$$

In the case of closed curves, the sum over $i < j < k$ will instead be over $0 \leq i, k \leq n$, and $j \in cw_n(i,k)$.

Let

$$
\begin{aligned}
Count_a^{(t)}(\langle X_{ik} \to Y_{ij} Z_{jk} \rangle) &= E_{T_a \sim \Delta_a^{(t)}}[\mathbf{1}[\langle X_{ik} \to Y_{ij} Z_{jk} \rangle]] \\
&= P_{T_a \sim \Delta_a^{(t)}}(\langle X_{ik} \to Y_{ij} Z_{jk} \rangle \in T_a) \\
Count_a^{(t)}(\langle X_{ii+1} \to \ell_{ii+1} \rangle) &= E_{T_a \sim \Delta_a^{(t)}}[\mathbf{1}[\langle X_{ii+1} \to \ell_{ii+1} \rangle]] \\
&= P_{T_a \sim \Delta_a^{(t)}}(\langle X_{ii+1} \to \ell_{ii+1} \rangle \in T_a)
\end{aligned}
$$

and let

$$Q^{(t)}_{a,i,j,k,[X\to YZ]}(\mathcal{G}) = \log P_{sub}(\langle X_{ik} \to Y_{ij}Z_{jk}\rangle) \cdot Count^{(t)}_a(\langle X_{ik} \to Y_{ij}Z_{jk}\rangle)$$
$$Q^{(t)}_{a,i,[X\to\ell]}(\mathcal{G}) = \log P_{sub}(\langle X_{ii+1} \to \ell_{ii+1}\rangle) \cdot Count^{(t)}_a(\langle X_{ii+1} \to \ell_{ii+1}\rangle).$$

3.1.2 Computing the Soft Counts

In this section, we show how to compute the soft counts from the inside and outside weights.

Firstly,

$$\begin{aligned}\Delta^{(t)}_a &= P(T_a \mid C_a, \mathcal{G}^{(t)}) \\ &= \frac{P(C_a, T_a \mid \mathcal{G}^{(t)})}{P(C_a \mid \mathcal{G}^{(t)})} \\ &= \frac{P(T_a \mid \mathcal{G}^{(t)})}{P(C_a \mid \mathcal{G}^{(t)})}.\end{aligned}$$

Definition 3.1.1. A *complete parse tree* is a parse tree with root node S_{0n}, and which has no unexpanded nodes. We will denote the set of complete $\mathcal{G}$-parse trees of C by $\mathbb{F}^C_{\mathcal{G}}$. We will denote the set of complete $\mathcal{G}$-parse trees of C which contain a node $\langle X_{ij} \to \lambda\rangle$ by $\mathbb{F}^C_{\mathcal{G}}(X_{ij}, \lambda)$.

Proposition 3.1.2. *Every parse tree $T \in \mathbb{F}^C_{\mathcal{G}}(X_{ij}, \lambda)$ can be obtained as*

$$T_{out}\Delta^{\cancel{X_{ij}}}_{(X_{ij},\lambda)}T_{in},$$

where $T_{out} \in \mathbb{O}^C_{\mathcal{G}}(X_{ij})$ and $T_{in} \in \mathbb{I}^C_{\mathcal{G}}(X_{ij}, \lambda)$.

Proof. If we remove the subtree rooted at the node $\langle X_{ij} \to \lambda\rangle$ and replace it with an unexpanded node X_{ij}, the resulting tree is an outer parse tree in $\mathbb{O}^C_{\mathcal{G}}(X_{ij})$. The subtree rooted at $\langle X_{ij} \to \lambda\rangle$ is an inner parse tree with root $\langle X_{ij} \to \lambda\rangle$. □

Observation 3.1.3. For open curves C,

$$P(\langle X_{ij} \to \lambda \rangle \in T \mid \mathcal{G}) = \sum_{T \in \mathbb{F}_{\mathcal{G}}^{C}(X_{ij}, \lambda)} W_{\mathcal{G}}(T)$$

which, by the previous proposition can be broken up as

$$\begin{aligned} &= \left(\sum_{T_{out} \in \mathbb{O}_{\mathcal{G}}^{C}(X_{ij})} W_{\mathcal{G}}(T_{out}) \right) \left(\sum_{T_{in} \in \mathbb{I}_{\mathcal{G}}^{C}(X_{ij}, \lambda)} W_{\mathcal{G}}(T_{in}) \right) \\ &= Outside_{\mathcal{G}}^{C}(X_{ij}) Inside_{\mathcal{G}}^{C}(X_{ij}, \lambda). \end{aligned}$$

For closed curves C,

$$\begin{aligned} P(\langle X_{ij} \to \lambda \rangle \in T \mid \mathcal{G}) &= \frac{1}{|C|} \sum_{T \in \mathbb{F}_{\mathcal{G}}^{C}(X_{ij}, \lambda)} W_{\mathcal{G}}(T) \\ &= \frac{1}{|C|} COutside_{\mathcal{G}}^{C}(X_{ij}) CInside_{\mathcal{G}}^{C}(X_{ij}, \lambda). \end{aligned}$$

In particular, for open curves, $P(C \mid \mathcal{G}) = Inside_{\mathcal{G}}^{C}(S_{0n})$, and for closed curves, $P(C \mid \mathcal{G}) = \frac{1}{|C|} \sum_{i=0}^{n-1} CInside_{\mathcal{G}}^{C}(S_{ii})$. Thus, for open curves,

$$Count_a(\langle X_{ij} \to \lambda \rangle) = \frac{Outside_{\mathcal{G}^{(t)}}^{C_a}(X_{ij}) Inside_{\mathcal{G}^{(t)}}^{C_a}(X_{ij}, \lambda)}{Inside_{\mathcal{G}^{(t)}}^{C_a}(S_{0n})},$$

and for closed curves,

$$Count_a(\langle X_{ij} \to \lambda \rangle) = \frac{COutside_{\mathcal{G}^{(t)}}^{C_a}(X_{ij}) CInside_{\mathcal{G}^{(t)}}^{C_a}(X_{ij}, \lambda)}{\frac{1}{|C_a|} \sum_{i=0}^{n-1} CInside_{\mathcal{G}^{(t)}}^{C_a}(S_{ii})},$$

To summarize,

$$
\begin{aligned}
Q^{(t)}(\mathcal{G}) = &\sum_a \sum_{i,j,k} \sum_{[X\to YZ]\in\mathcal{R}} Q^{(t)}_{a,i,j,k,[X\to YZ]}(\mathcal{G}) \\
&+ \sum_a \sum_i \sum_{[X\to\ell]\in\mathcal{R}} Q^{(t)}_{a,i,[X\to\ell]}(\mathcal{G}) \\
&+ \log P(\mathcal{G})
\end{aligned}
$$

$$
\begin{aligned}
Q^{(t)}_{a,i,j,k,[X\to YZ]}(\mathcal{G}) &= \log P_{sub}(\langle X_{ik} \to Y_{ij} Z_{jk}\rangle | \mathcal{G}) \cdot Count^{(t)}_a(\langle X_{ik} \to Y_{ij} Z_{jk}\rangle) \\
Q^{(t)}_{a,i,[X\to\ell]}(\mathcal{G}) &= \log P_{sub}(\langle X_{ii+1} \to \ell_{ii+1}\rangle | \mathcal{G}) \cdot Count^{(t)}_a(\langle X_{ii+1} \to \ell_{ii+1}\rangle)
\end{aligned}
$$

$$
Count_a(\langle X_{ij} \to \lambda\rangle) = \frac{Outside^{C_a}_{\mathcal{G}^{(t)}}(X_{ij}) Inside^{C_a}_{\mathcal{G}^{(t)}}(X_{ij}, \lambda)}{Inside^{C_a}_{\mathcal{G}^{(t)}}(S_{0n})},
$$

for open curves, and

$$
Count_a(\langle X_{ij} \to \lambda\rangle) = \frac{COutside^{C_a}_{\mathcal{G}^{(t)}}(X_{ij}) CInside^{C_a}_{\mathcal{G}^{(t)}}(X_{ij}, \lambda)}{\frac{1}{|C_a|} \sum_{i=0}^{n-1} CInside^{C_a}_{\mathcal{G}^{(t)}}(S_{ii})},
$$

for closed curves.

3.1.3 The M Step

We want to find the grammar parameters that maximize $Q^{(t)}(\mathcal{G})$. We have two classes of parameters: the multinomial distributions ρ_X, and the midpoint distributions $\mu_{X\to YZ}$. We will consider each in turn.

3.1.4 Learning Multinomial Distributions

A multinomial distribution is a distribution over a finite alphabet $a_1, \ldots, a_k$, where $P(a_i) = p_i$. Learning a multinomial distribution from independent samples $x_1, \ldots, x_n$ is commonly done by maximizing the posterior probability

$$\begin{aligned}\vec{p} = (\widehat{p_1}, \ldots, \widehat{p_k}) &= \operatorname*{argmax}_{\vec{p}} P(\vec{p} \mid x_1, \ldots, x_n) \\ &= \operatorname*{argmax}_{\vec{p}} P(x_1, \ldots, x_n; \vec{p}) \cdot P(\vec{p}),\end{aligned}$$

where $P(\vec{p})$ is a prior distribution over the space of possible $(p_1, \ldots, p_k)$.

The most common choice of prior is the Dirichlet distribution, parameterized by $\alpha_1, \ldots, \alpha_k > 0$:

$$\mathrm{Dir}(p_1, \ldots, p_k; \alpha_1, \ldots, \alpha_k) = \frac{\Gamma(\sum \alpha_i)}{\prod \Gamma(\alpha_i)} \prod_{i=1}^{k} p_i^{\alpha_i - 1},$$

where

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} \, \mathrm{d}t.$$

This is because, when all $\alpha_i \geq 1$, the maximum a posteriori estimate of the $\{p_i\}$ has the simple form

$$\widehat{p_i} = \frac{c_i + \alpha_i - 1}{n + \sum \alpha_i - k},$$

where

$$c_i = \#\{j : x_j = i\}.$$

More generally, the maximum a posteriori estimate has the form

$$\widehat{p_i} = \begin{cases} \frac{c_i + \alpha_i - 1}{n + \sum_{i \in I} \alpha_i - |I|} & c_i + \alpha_i \geq 1 \\ 0 & \text{otherwise} \end{cases},$$

where

$$I = \{i : c_i + \alpha_i \geq 1\}.$$

Note that results still hold when the counts c_i are real rather than integer. This is because the proofs rely solely on the log-likelihood having the form $\sum c_i \log p_i$. In this case, the c_i are referred to as “soft counts”.

When the α_i are greater than one, this prior smooths the estimated distribution $\vec{p}$ towards the uniform distribution. When the α_i are less than one, this prior makes the distribution sparser than the maximum likelihood estimate.

When estimating the production probabilities in a grammar, a sparsifying prior is generally recommended. One intuition for this is that we are trying to recover a relatively sparse set of rules from the set of all possible rules; in order to avoid missing a rule, we must give some weight to (and thus generate observations of) rules that will ultimately be discarded. In Johnson et al. [2007], a Dirichlet prior with $\alpha_i = \frac{1}{1000}$ was found to work well for training context-free grammars for natural languages, although α_i between $\frac{1}{100}$ and $\frac{1}{10{,}000}$ were found to work about as well.

In our case, $\rho_X([X \to \lambda])$ is a multinomial distribution, and we have soft counts $c_\lambda = \sum_{a,i,j} Count_a(\langle X_{ij} \to \lambda \rangle)$.

3.1.5 Learning Watson distributions

First, we choose priors. Let $X \to YZ$ be a rule of G. We set $\mu_{X\to YZ}$ equal to a Watson distribution with random mode μ and random concentration κ, where

$$\kappa \sim Gamma(\sigma, \theta = \bar{\kappa}/\sigma)$$

$$\mu \sim Watson(\mu_0, \alpha_{mp}\kappa)$$

where σ is the shape parameter of the Gamma distribution, and $\bar{\kappa} = \alpha_\kappa s$.

Currently $\sigma = 5$, $\alpha_\kappa = 4000.0$. α_{mp} is a parameter dictating the balance between prior and observation. We choose μ_0 to correspond to $\begin{pmatrix} p & q & r \end{pmatrix}$ with q being the mean of p and r.

We wish to update our estimate of the distribution $\mu_{X\to YZ}$. Since we are using a Watson distribution, this means estimating the parameters μ and κ. Let $z_{a,ijk}$ be the midpoint observed in curve a, when we put $C_a[j]$ in coordinates relative to $C_a[i]$ and $C_a[k]$. The function we are trying to maximize, $Q^{(t)}$, depends on $\mu_{X\to YZ}$ as a sum of log probabilities:

$$\sum_{a,i,j,k} Count_a^{(t)}(\langle X_{ik} \to Y_{ij}Z_{jk}\rangle) \log \mu_{X\to YZ}(z_{a,ijk})$$
$$+ \log Gamma(\kappa; \sigma, \theta) + \log Watson(\mu; \mu_0, \alpha_{mp}\kappa).$$

Thus, the *Count* values are the soft counts we use to weight the $z_{a,ijk}$. For simplicity, we will simply say that we have observations $z_1, \ldots, z_n$ with corresponding weights $c_1, \ldots, c_n$.

Let $\gamma(\kappa; \sigma, \theta) = \log Gamma(\kappa; \sigma, \theta)$, where

$$Gamma(x; \sigma, \theta) = \frac{1}{\theta^\sigma} \frac{1}{\Gamma(\sigma)} x^{\sigma-1} e^{-\frac{x}{\theta}}.$$

Let $c(\kappa)$ be the normalizing constant of the Watson distribution.

$$\begin{aligned}
P(\mu, \kappa \mid z_1, \ldots, z_m) &\propto P(\mu, \kappa) P(z_1, \ldots, z_m \mid \mu, \kappa) \\
&= \exp[\alpha_{mp} \kappa \mu^* \mu_0 \mu_0^* \mu - \log c(\alpha_{mp} \kappa) + \gamma(\kappa; \sigma, \theta) \\
&\quad + \kappa \sum_i \mu^* z_i z_i^* \mu - m \log c(\kappa) \\
&= \exp[\kappa \mu^* (\alpha_{mp} \mu_0 \mu_0^* + \sum z_i z_i^*) \mu - \log c(\alpha_{mp} \kappa)] \\
&\quad - m \log c(\kappa) + \gamma(\kappa; \sigma, \theta)] \\
\log P(\mu, \kappa \mid z_1, \ldots, z_m) &= c_{norm} + \kappa \mu^* (\alpha_{mp} \mu_0 \mu_0^* + \sum z_i z_i^*) \mu - \log c(\alpha_{mp} \kappa) \\
&\quad - m \log c(\kappa) + \gamma(\kappa; \sigma, \theta) \\
&= c_{norm} + \kappa \mu^* A \mu - \log c(\alpha_{mp} \kappa) - m \log c(\kappa) + \gamma(\kappa; \sigma, \theta),
\end{aligned}$$

Let Q be the terms in $Q^{(t)}$ which depend on $\mu_{X \to YZ}$.

$$\begin{aligned}
Q(\mu, \kappa) &= \log P_{prior}(\mu, \kappa) + \sum_i c_i \log P(z_i \mid \mu, \kappa) \\
&= \alpha_{mp} \kappa \mu^* \mu_0 \mu_0^* \mu - \log c(\alpha_{mp} \kappa) + \gamma(\kappa; \sigma, \theta) + \kappa \sum_i c_i \mu^* z_i z_i^* \mu - (\sum_i c_i) \log c(\kappa) \\
&= \kappa \mu^* (\alpha_{mp} \mu_0 \mu_0^* + \sum_i c_i z_i z_i^*) \mu - \log c(\alpha_{mp} \kappa) - (\sum_i c_i) \log c(\kappa) + \gamma(\kappa; \sigma, \theta) \\
&= c_{norm} + \kappa \mu^* (\alpha_{mp} \mu_0 \mu_0^* + \sum_i c_i z_i z_i^*) \mu - \log c(\alpha_{mp} \kappa) - (\sum_i c_i) \log c(\kappa) \\
&\quad + \gamma(\kappa; \sigma, \theta) \\
&= c_{norm} + \kappa \mu^* A \mu - \log c(\alpha_{mp} \kappa) - m \log c(\kappa) + \gamma(\kappa; \sigma, \theta),
\end{aligned}$$

where $A = \alpha_{mp} \mu_0 \mu_0^* + \sum c_i z_i z_i^*$ and $m = \sum_i c_i$. Since μ only appears in the term $\kappa \mu^* A \mu$, and A has non-negative eigenvalues, it is clear that we maximize the posterior by choosing μ to be the dominant eigenvector of A. Let $\lambda_{\max}$ be the associated eigenvalue. Then

$$\log P(\mu, \kappa \mid z_1, \ldots, z_m) = c_{norm} + \kappa\lambda_{\max}(A) - \log c(\alpha_{mp}\kappa) - m \log c(\kappa) + \gamma(\kappa; \sigma, \theta)$$

Taking partial derivatives, and using the approximation $c(\kappa) = 2\pi^2 \frac{1}{\kappa} e^{\kappa}$, we arrive at the approximate solution

$$\frac{\partial \log P(\mu, \kappa \mid, z_1, \ldots, z_m)}{\partial \kappa} = \lambda_{max} - \alpha_{mp} - m + \frac{1+m}{\kappa} + \frac{\sigma - 1}{\kappa} - \frac{\sigma}{\bar{\kappa}} = 0$$

$$\widehat{\kappa} = \frac{\sigma + m}{\frac{\sigma}{\bar{\kappa}} + m + \alpha_{mp} - \lambda_{max}}$$

$$\widehat{\mu} = \text{ dominant eigenvector of } A, \text{ where } A = \alpha_{mp}\mu_0\mu_0^* + \sum z_i z_i^*$$

Examining this solution, we see that our estimate of the midpoint is similar to the maximum likelihood estimate of a Watson distribution, but is adapted to contain α_{mp} artificial observations at the prior mode.

The posterior estimate of the concentration will be at a maxmimum when all z_i are identical and equal to μ_0. In this case, λ_{max} will be $m + \alpha_{mp}$, and $\widehat{\kappa} = \bar{\kappa}\frac{\sigma+n}{\sigma}$. Thus, our concentration will increase essentially linearly with the number of examples, but will never be infinite.

In general, the z_i and μ_0 will be more spread out, causing A to have a lower dominant eigenvalue, and lowering the concentration.

3.1.6 Examples of Learning Watson distributions

In figures 3.1 through 3.5, we demonstrate that we can learn the mode and concentration parameters of Watson distributions from training data. In each experiment, we select a random triangle (by using a Gaussian in Bookstein coordinates). We then draw 20 samples from

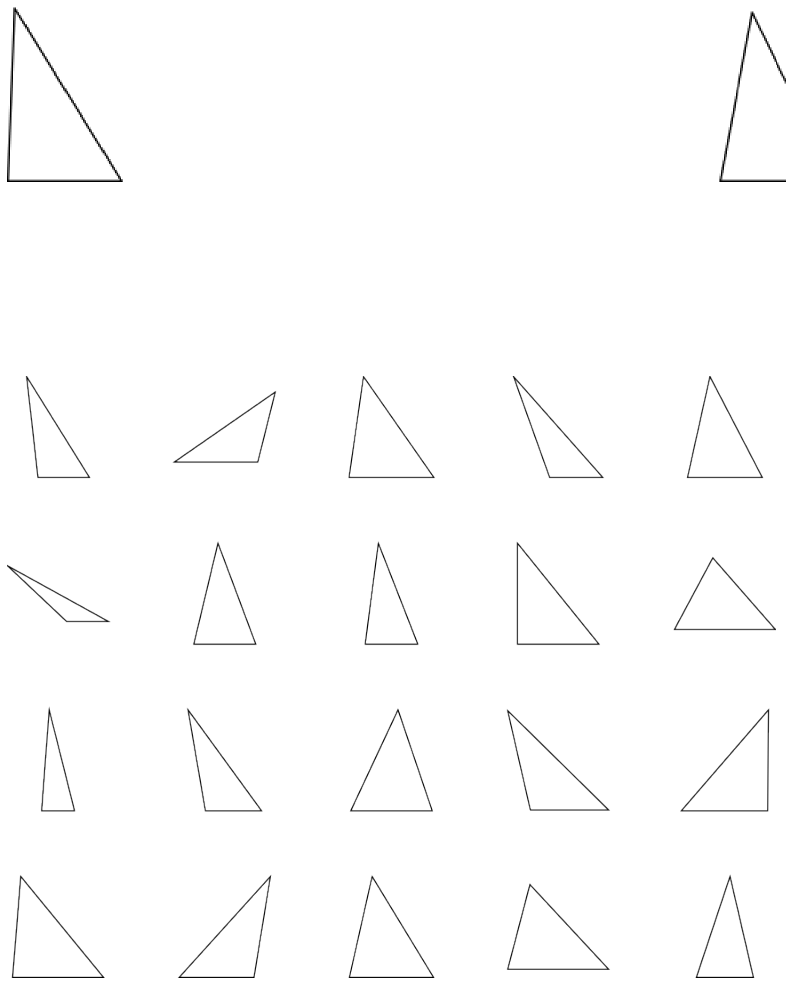


Figure 3.1: Experimenting with the Watson distribution, part 1. In the first row, the original triangle T on the left, and the mode of the estimated Watson distribution on the right. Subsequent rows are samples from Watson$(T, 30.00)$. The estimated concentration was 63.48.

the Watson distribution centered at this triangle (using 30 for the concentration parameter of the Watson). We then reestimate the Watson distribution from the samples. This is a less noisy version of the learning task that EM faces when refitting the midpoint distributions of a grammar from 20 samples.

We also show the results of fitting a Watson distribution with differing numbers of samples in Table 3.6. This demonstrates that the estimated concentration approaches the correct value as the number of samples increases.

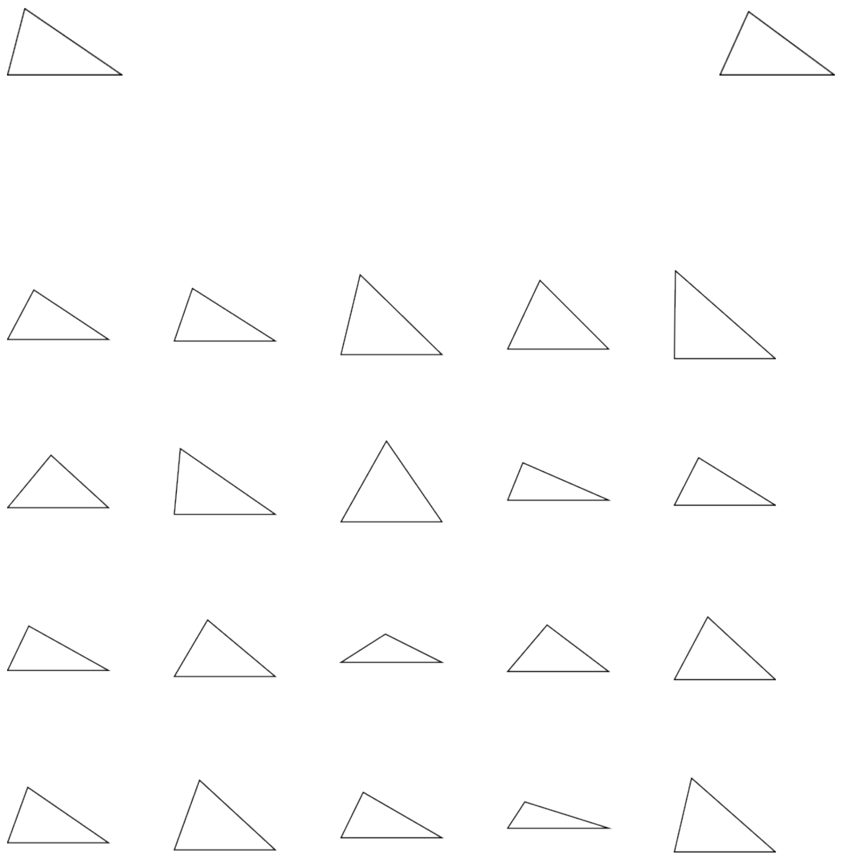


Figure 3.2: Experimenting with the Watson distribution, part 2. In the first row, the original triangle T on the left, and the mode of the estimated Watson distribution on the right. Subsequent rows are samples from Watson$(T, 30.00)$. The estimated concentration was 82.09.

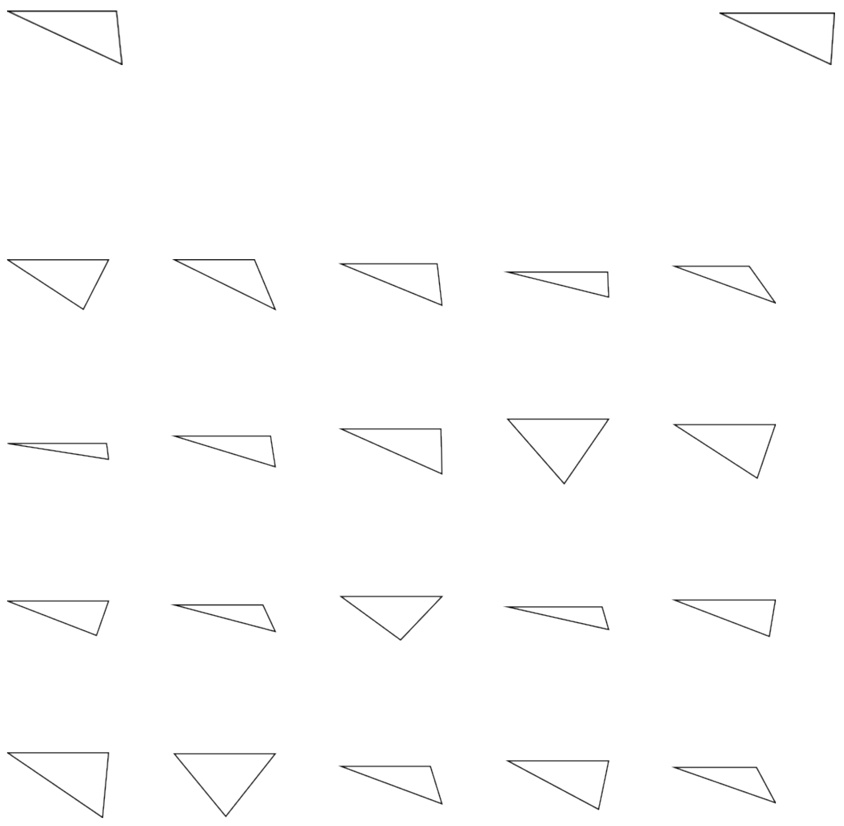


Figure 3.3: Experimenting with the Watson distribution, part 3. In the first row, the original triangle T on the left, and the mode of the estimated Watson distribution on the right. Subsequent rows are samples from Watson$(T, 30.00)$. The estimated concentration was 69.92.

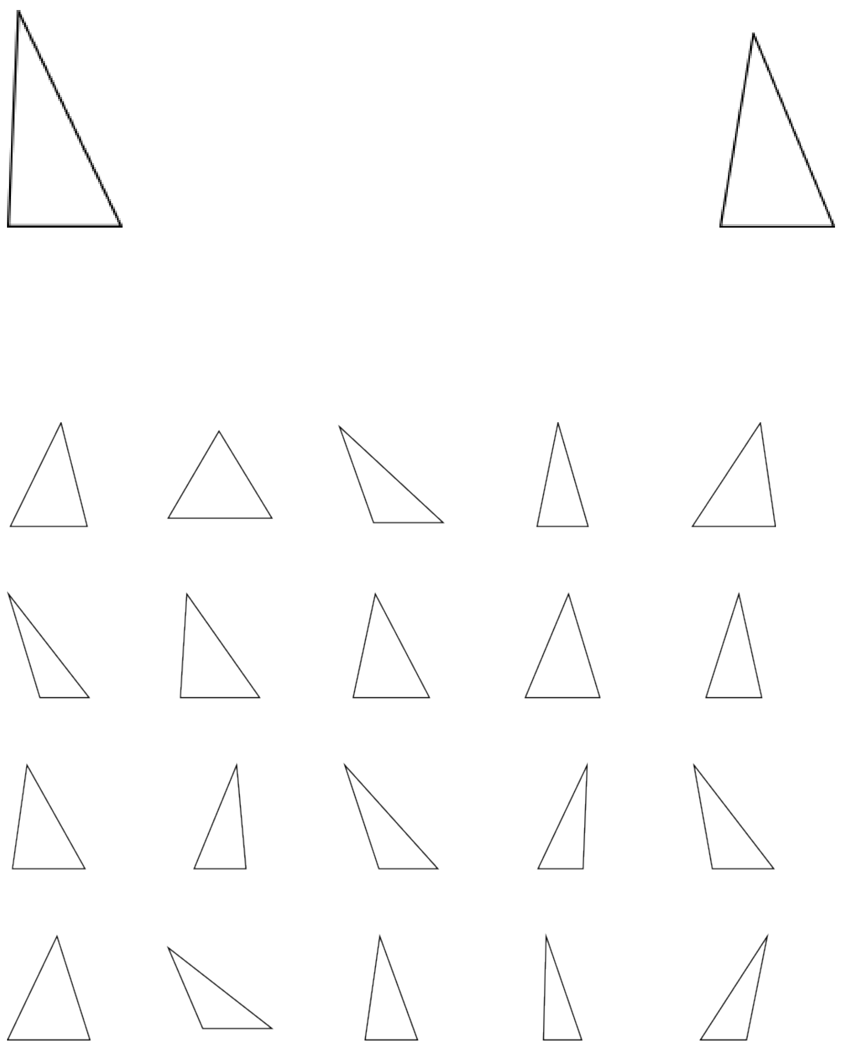


Figure 3.4: Experimenting with the Watson distribution, part 4. In the first row, the original triangle T on the left, and the mode of the estimated Watson distribution on the right. Subsequent rows are samples from Watson$(T, 30.00)$. The estimated concentration was 77.10.

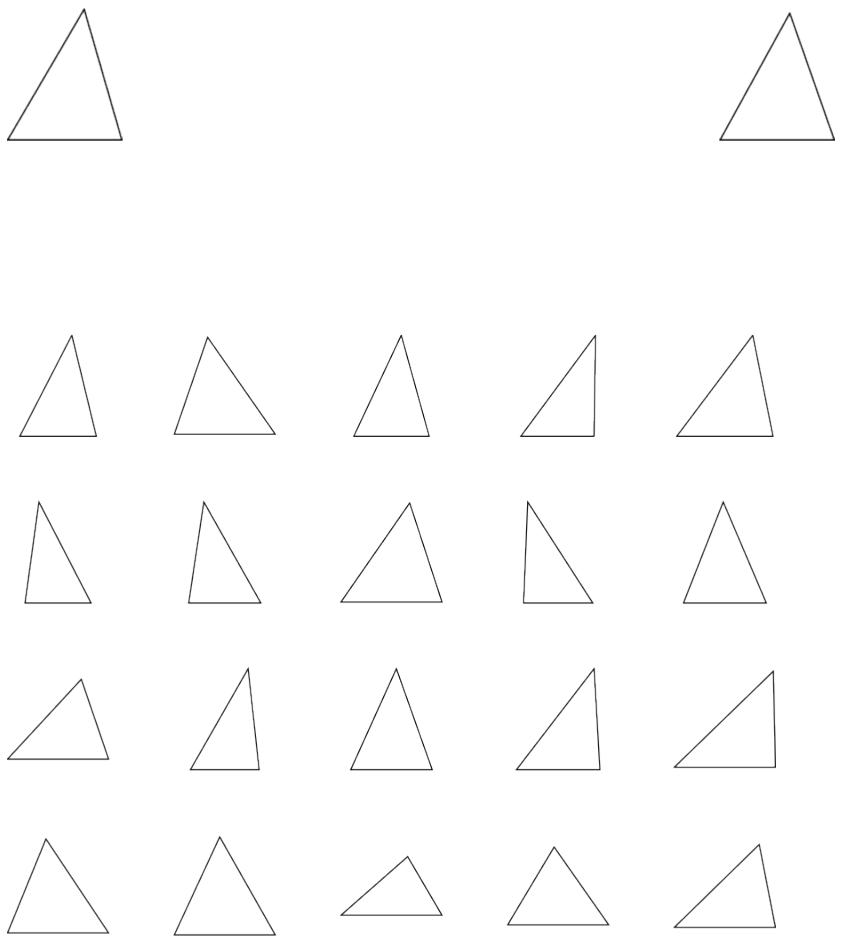


Figure 3.5: Experimenting with the Watson distribution, part 5. In the first row, the original triangle T on the left, and the mode of the estimated Watson distribution on the right. Subsequent rows are samples from Watson$(T, 30.00)$. The estimated concentration was 79.39.

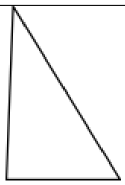
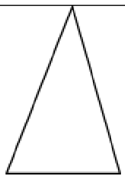
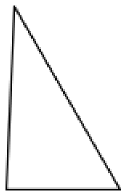
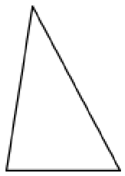
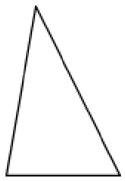
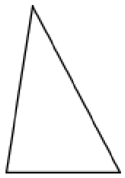
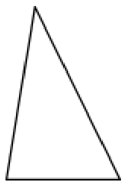
number of samples	mean	concentration
(true)		30.00
3		87.55
10		79.70
30		62.15
100		42.28
300		32.47
1000		29.89

Figure 3.6: Fitting the Watson distribution with different numbers of samples.

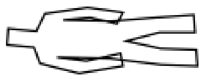


Figure 3.7: The example curve used to initialize grammars.

3.2 Setup for Learning Experiments

For each experiment in this section, we build an initial grammar from a single example curve (shown in Figure 3.7), train it on the training curves (shown in Figure 3.8), and then evaluate it by computing the cross-entropy on the validation curves (shown in Figure 3.9). We show samples from each model in order to demonstrate what has been learned. We want models to generate reasonable-looking human silhouettes, and ideally generate some that differ significantly from any training example. For instance, we would like the model to learn that limbs can move independently of one another.

For all but one of the experiments, we repeat the experiment with three different starting structures, which are shown in Figures 3.10, 3.11, and 3.12 as decompositions of the example curve.

Unless otherwise noted, these experiments use Watson distributions and a $Gamma(\sigma = 10^6, \overline{\kappa} = 1000)$ prior on the concentration of the Watson distributions.

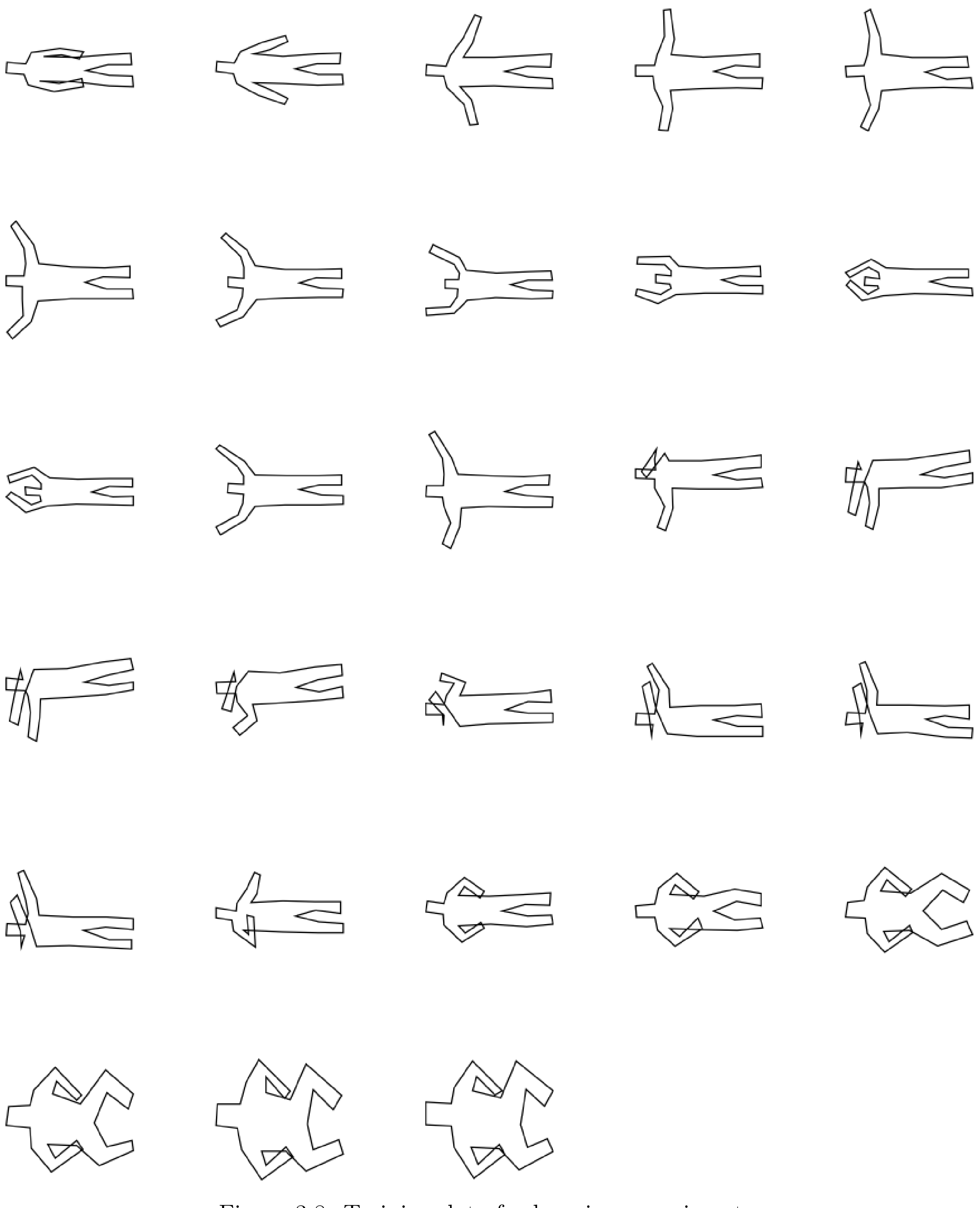


Figure 3.8: Training data for learning experiments.

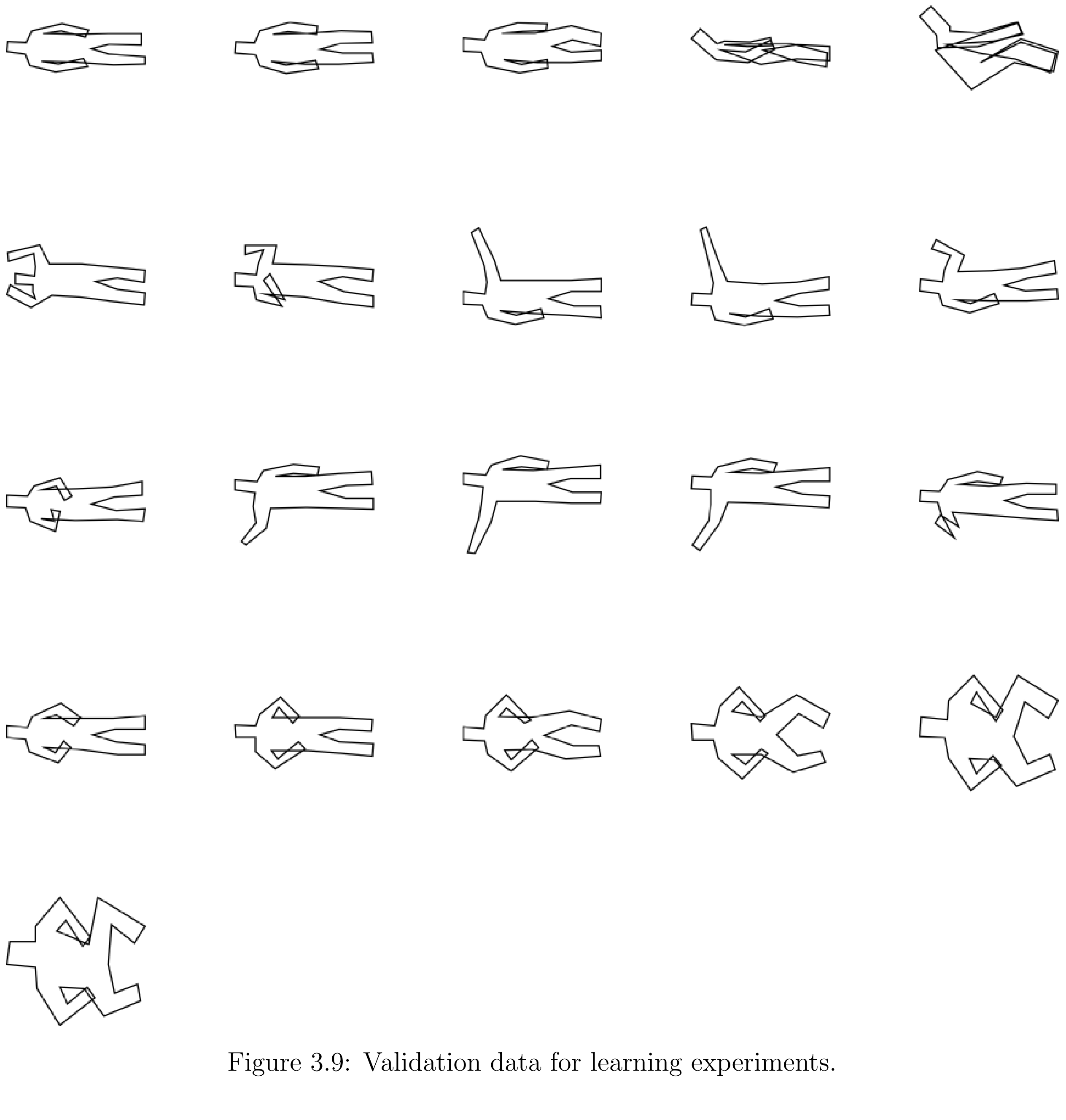


Figure 3.9: Validation data for learning experiments.

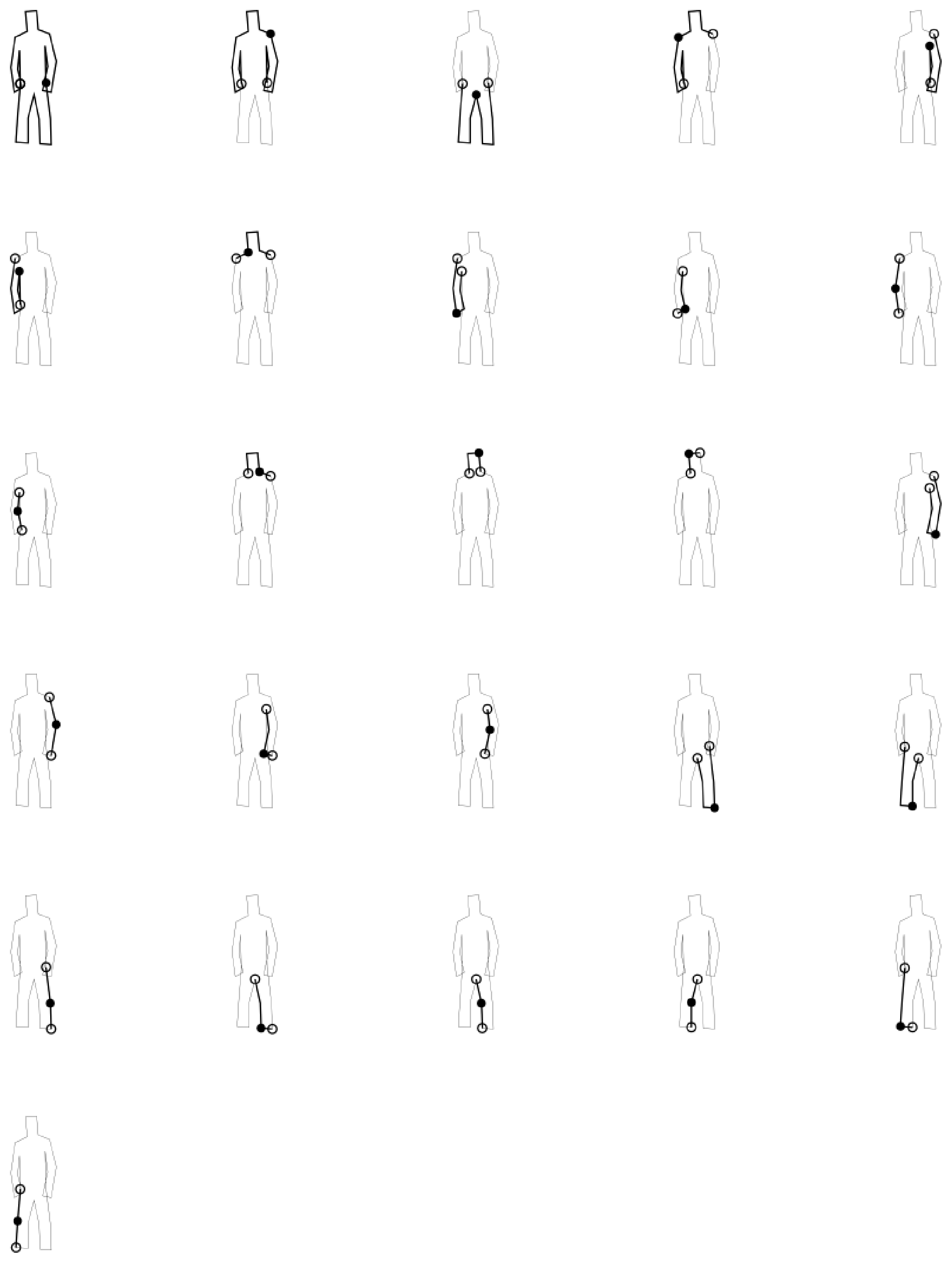


Figure 3.10: One of three starting structures used in learning experiments. Each bold curve is a symbol in the grammar, and each division of a subcurve corresponds to one production of the grammar.

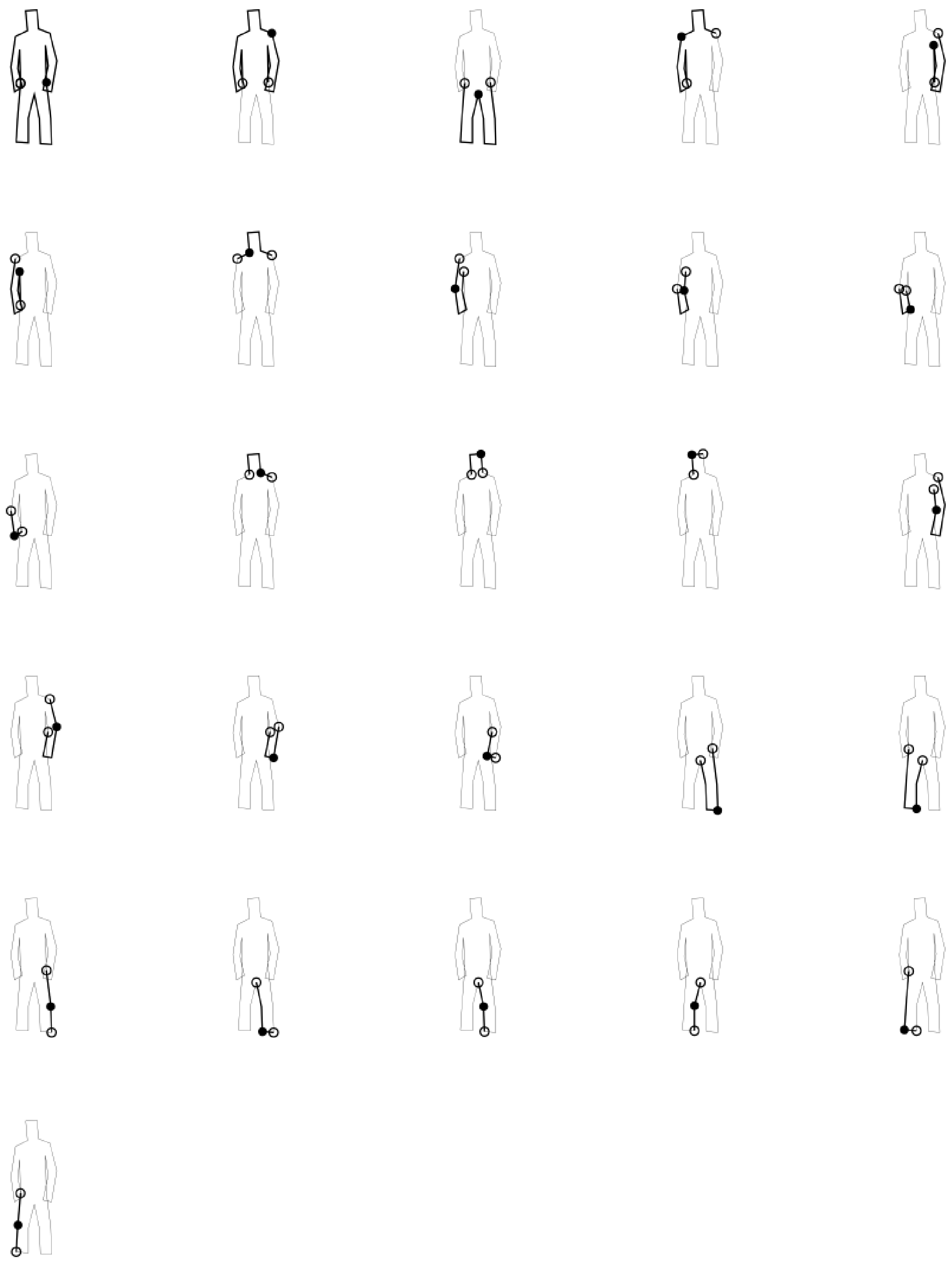


Figure 3.11: One of three starting structures used in learning experiments. Each bold curve is a symbol in the grammar, and each division of a subcurve corresponds to one production of the grammar.

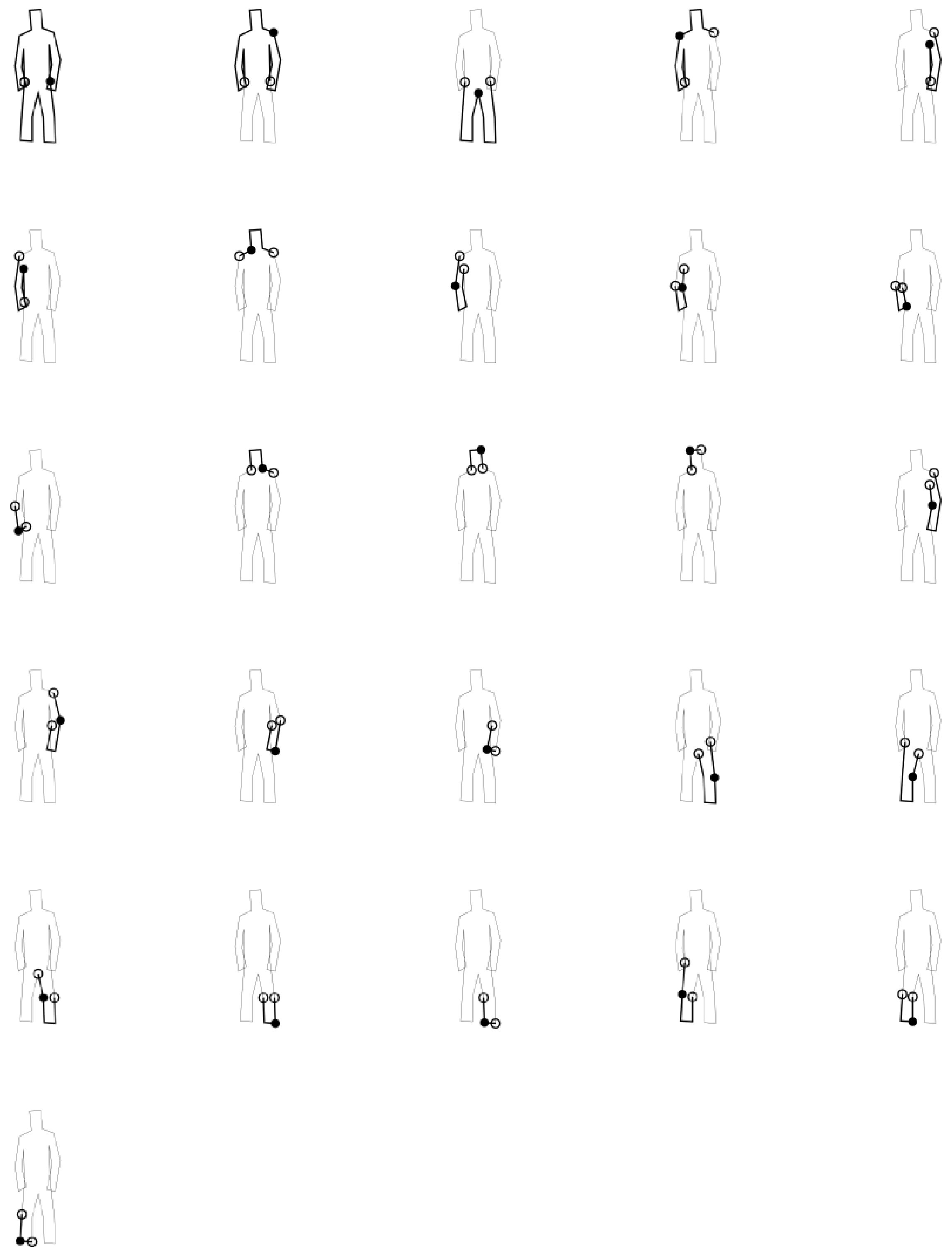


Figure 3.12: One of three starting structures used in learning experiments. Each bold curve is a symbol in the grammar, and each division of a subcurve corresponds to one production of the grammar.

3.3 Simple Tuning of hand-built grammar with curves of constant length

First, we use EM to train an initial grammar that has no choices: each symbol is on the left hand side of a single rule, and the only randomness of the model is in the midpoint distributions. Our grammar is produced from a hand-chosen decomposition. Samples from the grammar after various numbers of rounds of training are shown below. Since we are using unimodal midpoint distributions, and our grammar has no choice, all the samples are slight variations on one particular mean shape. The mean shape chosen is a very reasonable one, although the arms are longer than is ideal.

We repeat this experiment three times with different initial grammatical structures, to show how the performance of the algorithm depends on our initial structure. Samples from the model learned are shown in Figures 3.13, 3.14, and 3.15. There is only a small difference between the samples from different structures.

Simple Tuning	21679.145584	14944.299535	15952.025953
Multiple Midpoints	7112.552734	4620.838965	5732.380757
Multi-level Correlations	8451.596984	6095.411852	7368.234718
Full Grammar	2807.826409		

Table 3.1: Cross-entropy scores for experiments. Each column represents a different starting structure.

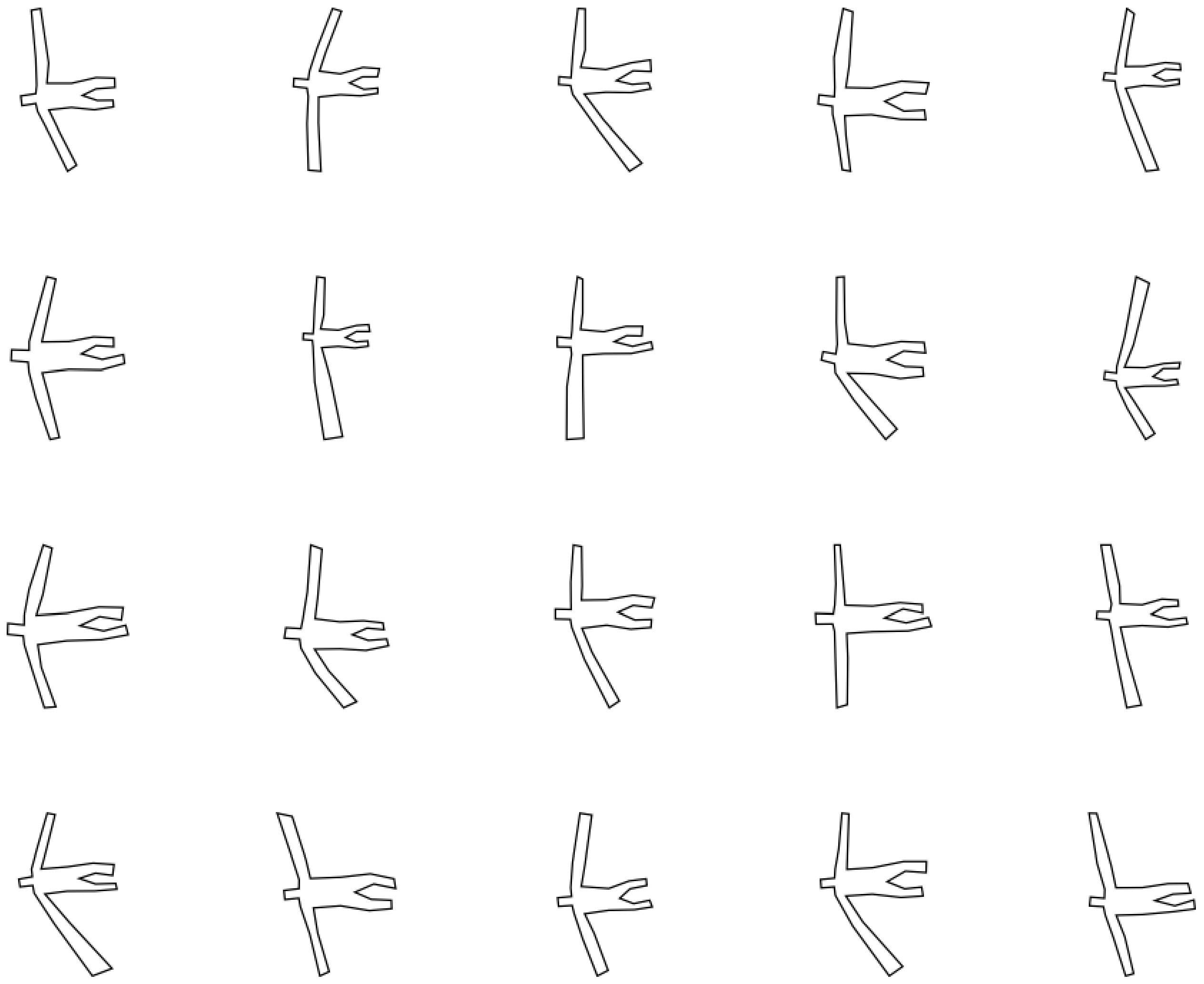


Figure 3.13: Samples from grammar after 30 rounds of EM, using a grammar based on structure 1. From simple tuning experiment.

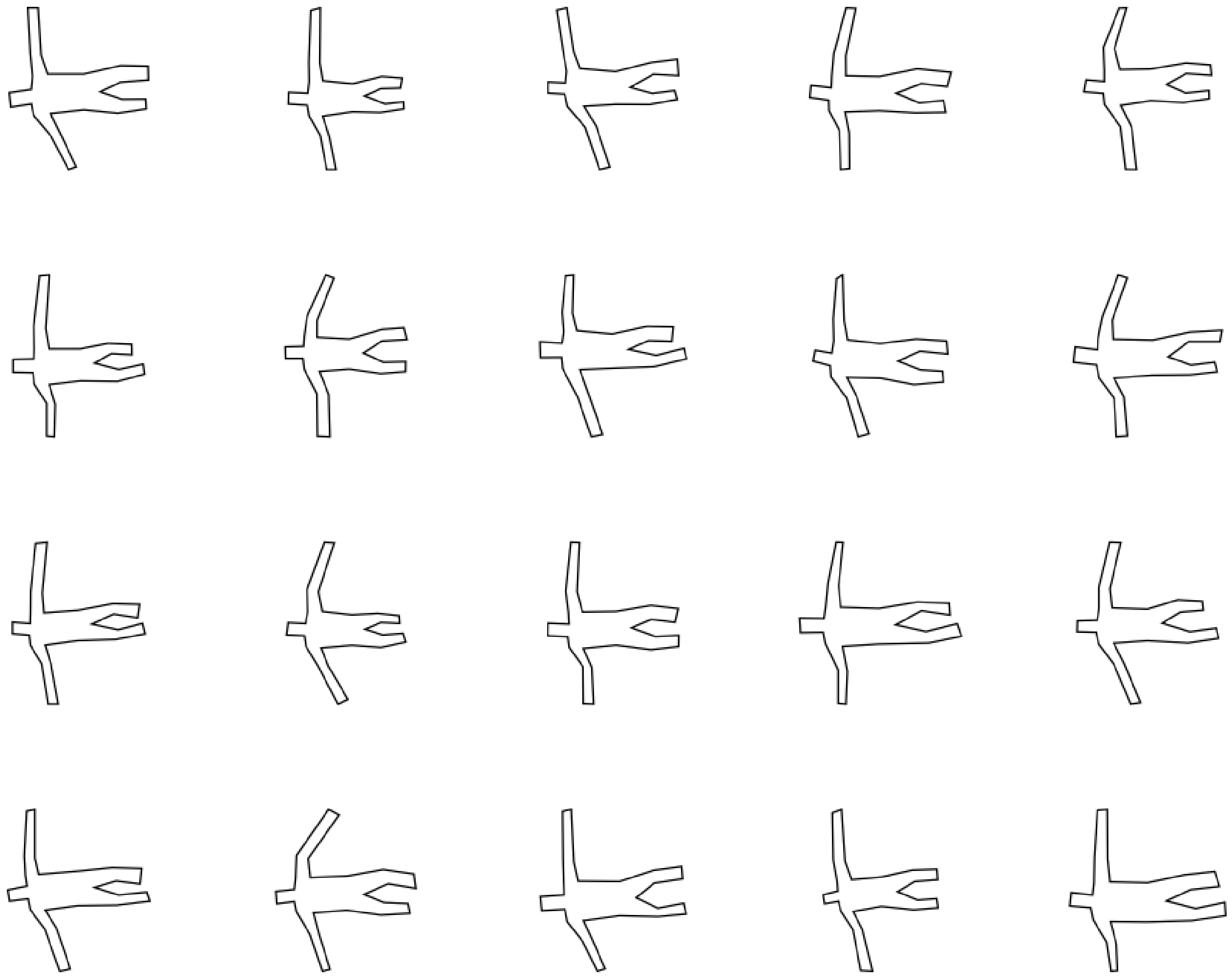


Figure 3.14: Samples from grammar after 30 rounds of EM, using a grammar based on structure 2. From simple tuning experiment.

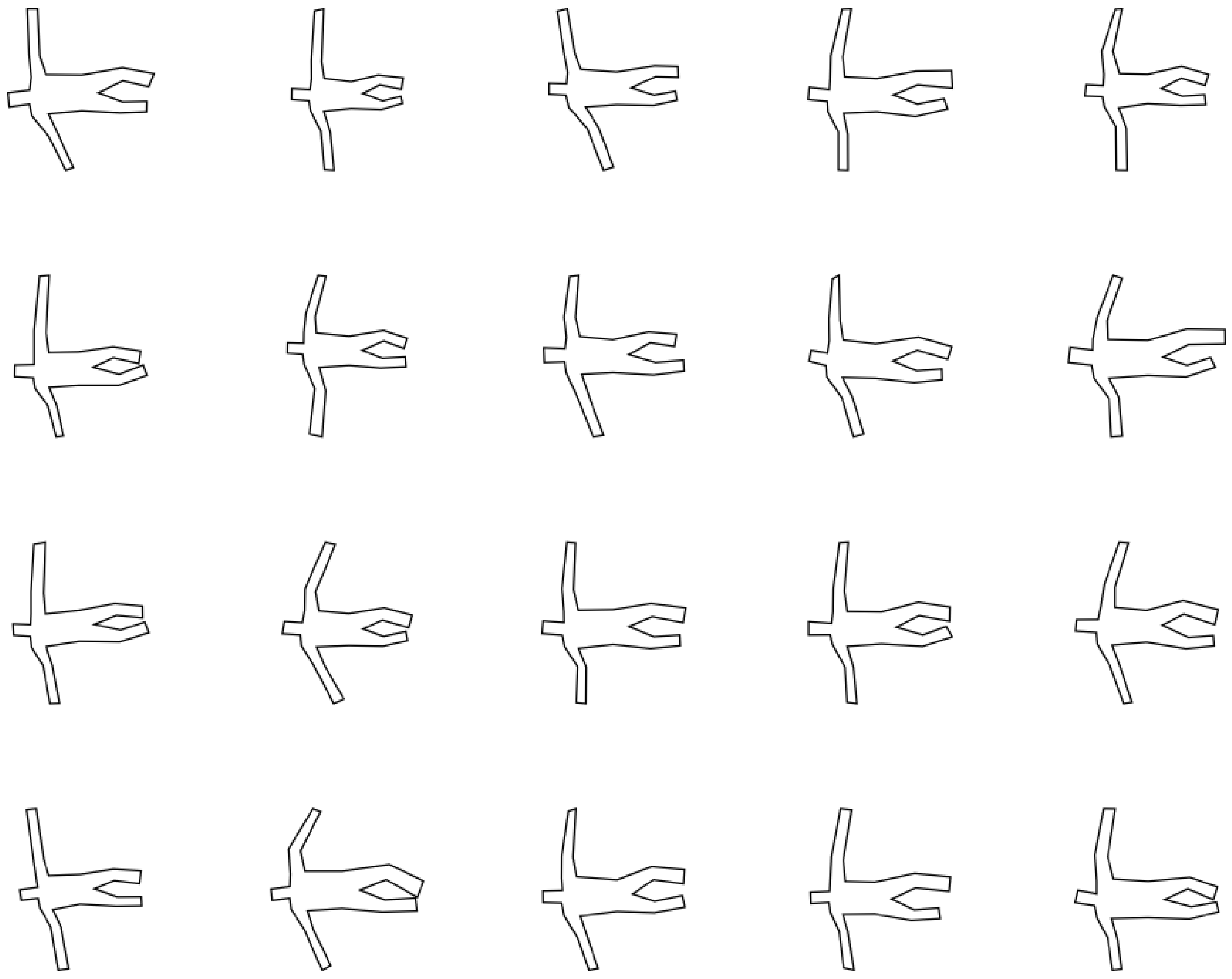


Figure 3.15: Samples from grammar after 30 rounds of EM, using a grammar based on structure 3. From simple tuning experiment.

3.4 Experimenting with different priors over concentration

In Figures 3.16, 3.17, and 3.18, we show the outcome from training with Watson distributions and a $Gamma(\sigma, \overline{\kappa} = 100)$ prior on the concentration of the Watson distributions, for $\sigma = 1, 1000, 10^6$. It can be seen that the samples become more realistic the higher we adjust the weight of the prior.

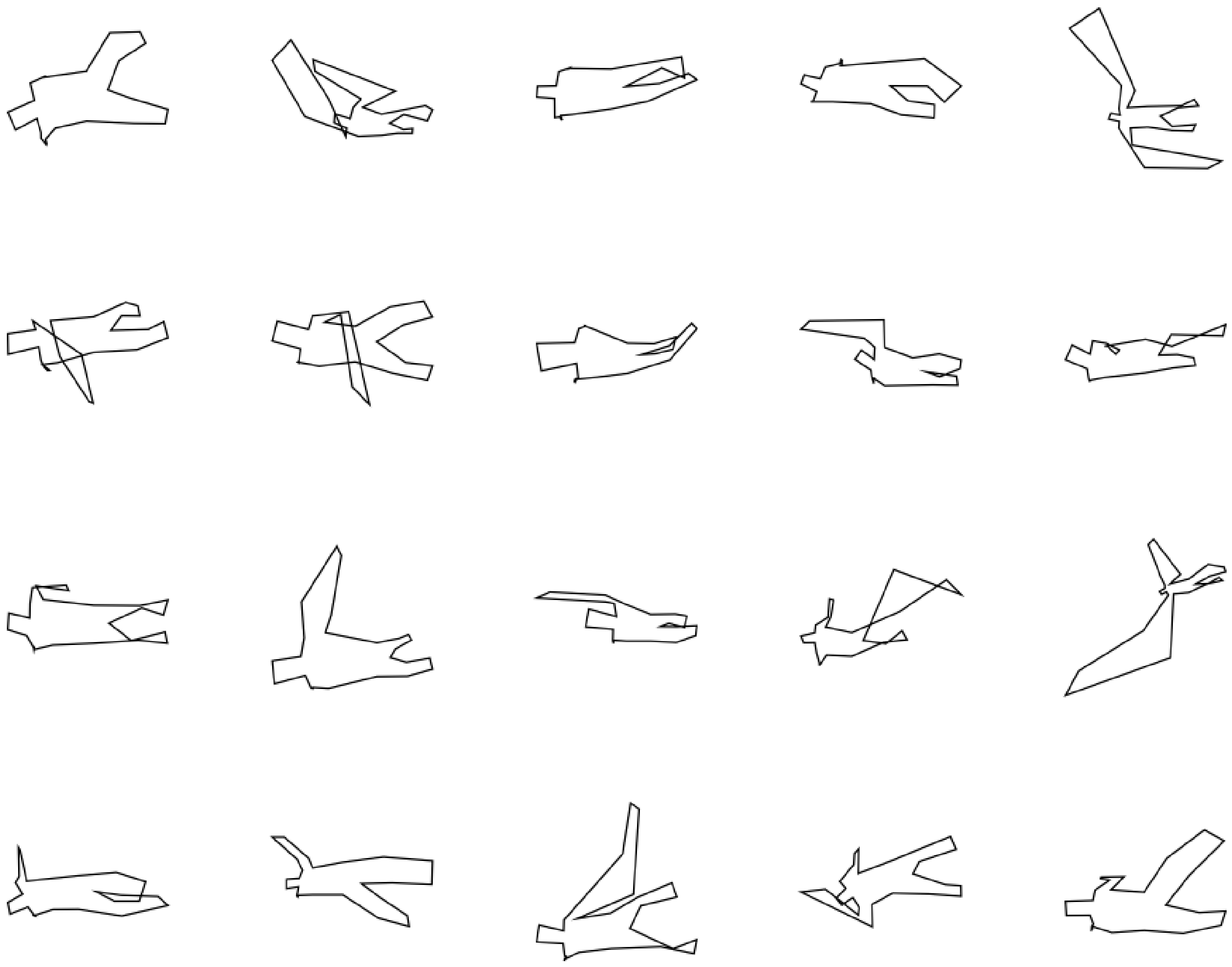


Figure 3.16: Samples from grammar after 30 rounds of EM, using $\sigma = 1$.

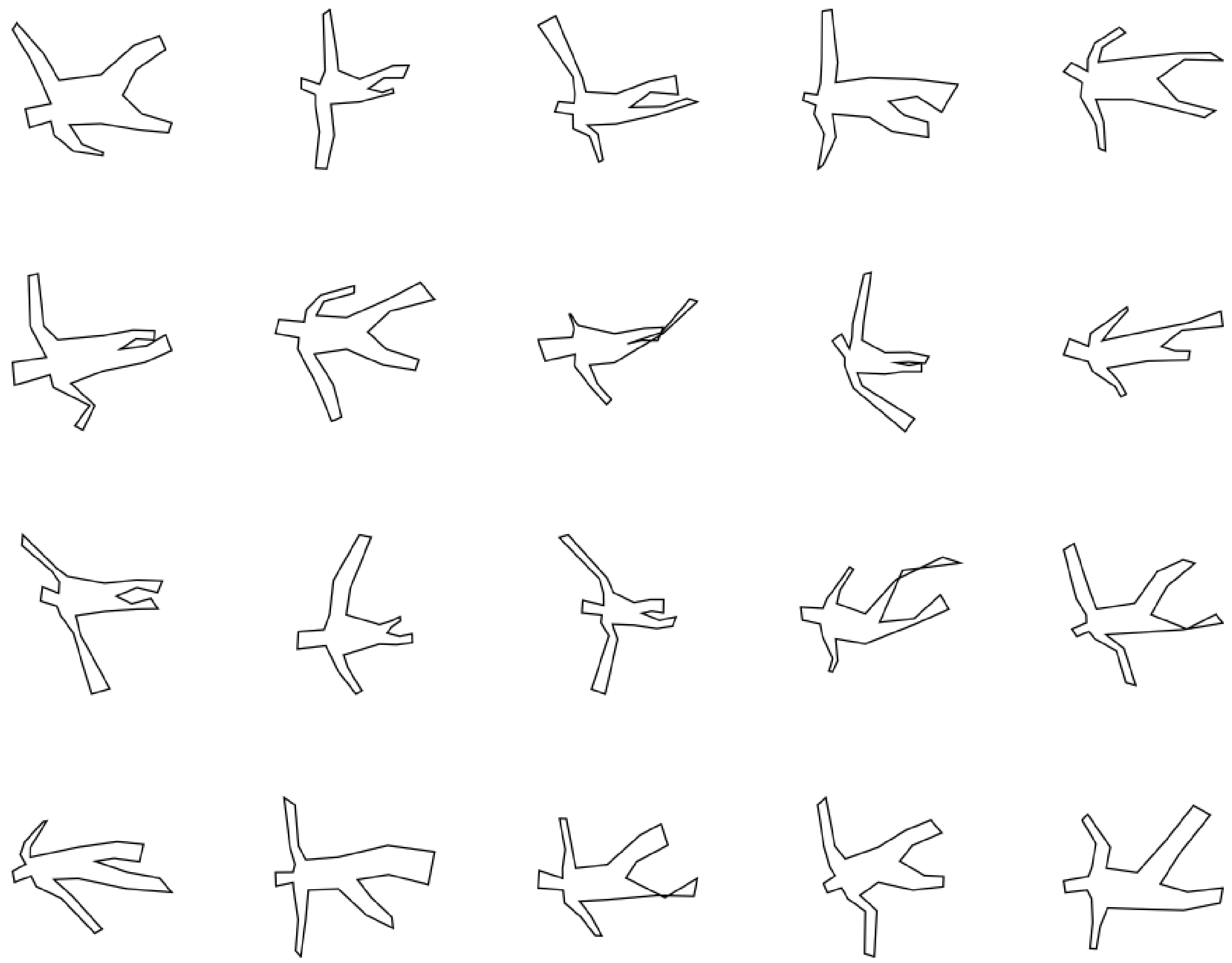


Figure 3.17: Samples from grammar after 30 rounds of EM, using $\sigma = 1000$.

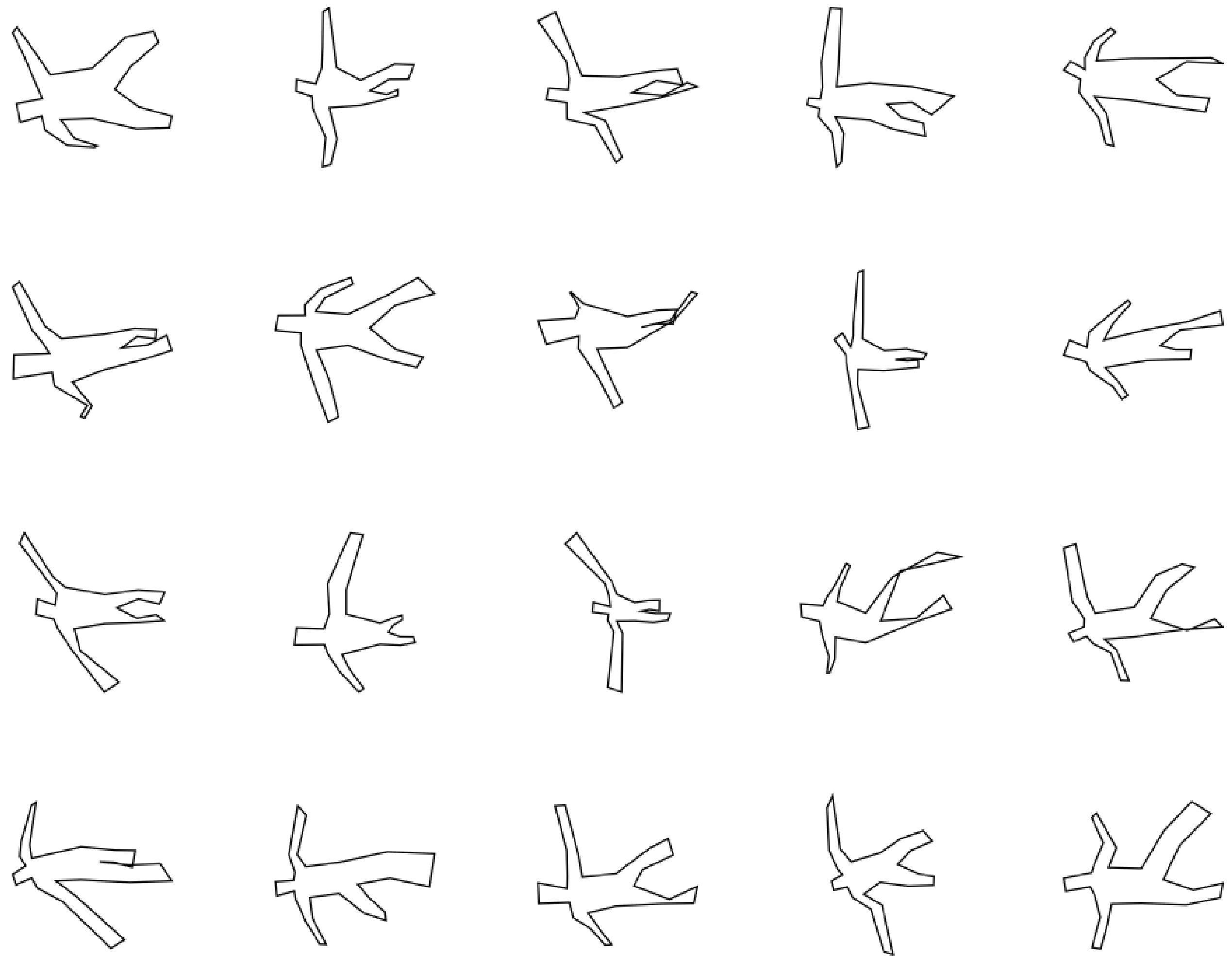


Figure 3.18: Samples from grammar after 30 rounds of EM, using $\sigma = 1000000$.

3.5 Multiple Midpoints, and Curves of Constant Length

In the next experiment, we enrich the grammar by adding in several copies (in this case, five) of each rule, with perturbed midpoints. We randomly perturb the mode of the midpoint distribution for each copy of the rule, in order to allow the different copies to be used to explain different data. (If we used the same midpoint for each copy of the rule, parses could use the rules interchangeably, and EM would never break this symmetry.) In Figures 3.19, 3.20, and 3.21, we show samples from the grammar built using each of the three starting structures. The samples show significantly more variation than in the previous experiment, but they also include many ill-formed and unlikely shapes. This is because the grammar cannot model correlations between levels, so that e.g., it cannot pick the location of the elbow based on the location of the hand, which means that the arm is often ill-formed and self-intersecting.

We repeat this grammar three times with different initial grammatical structures, to show how the performance of the algorithm depends on our initial structure. We show the log-likelihoods of the validation data after training in Table 3.1.

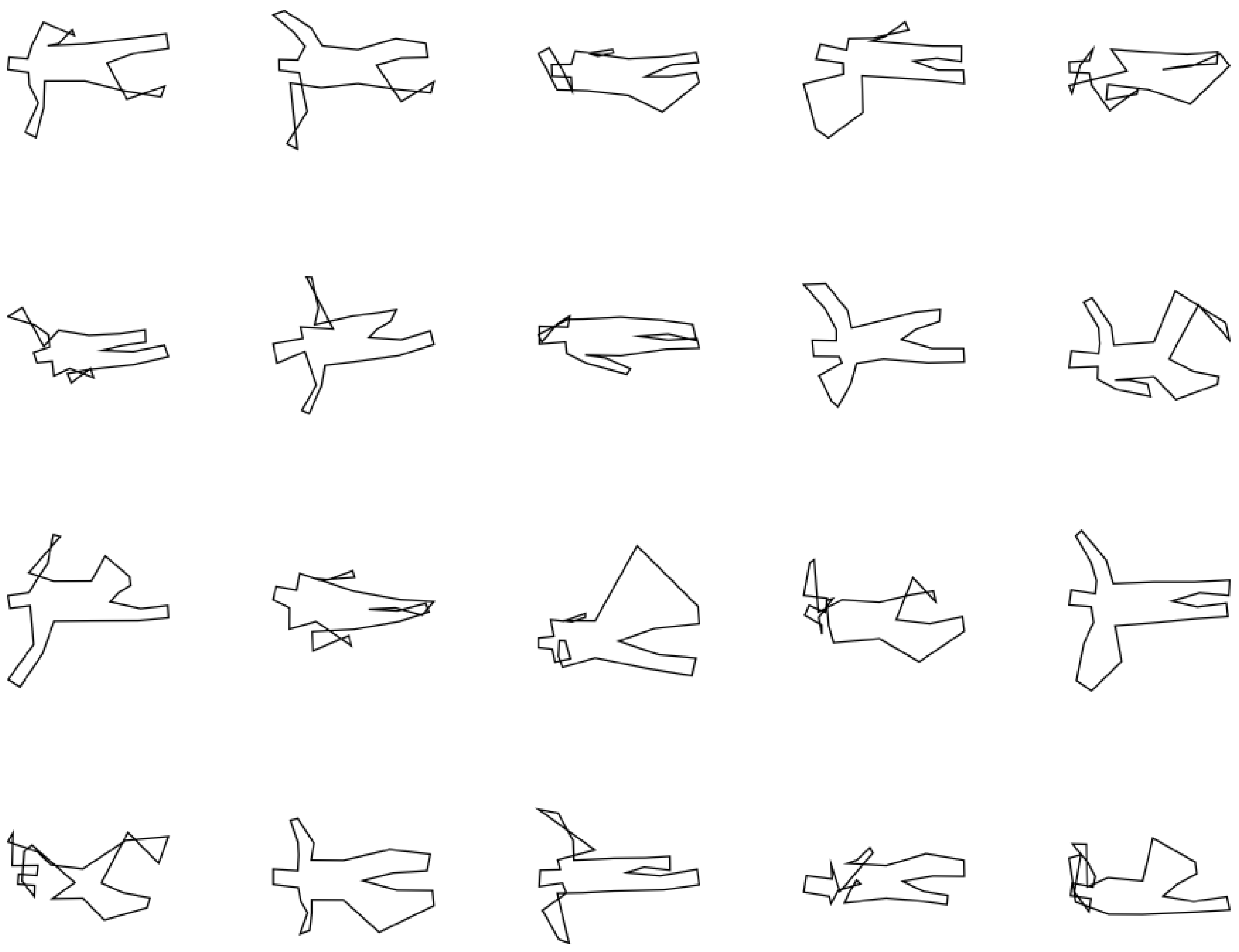


Figure 3.19: Samples from grammar after 30 rounds of EM, using structure 1. From multiple tuning experiment.

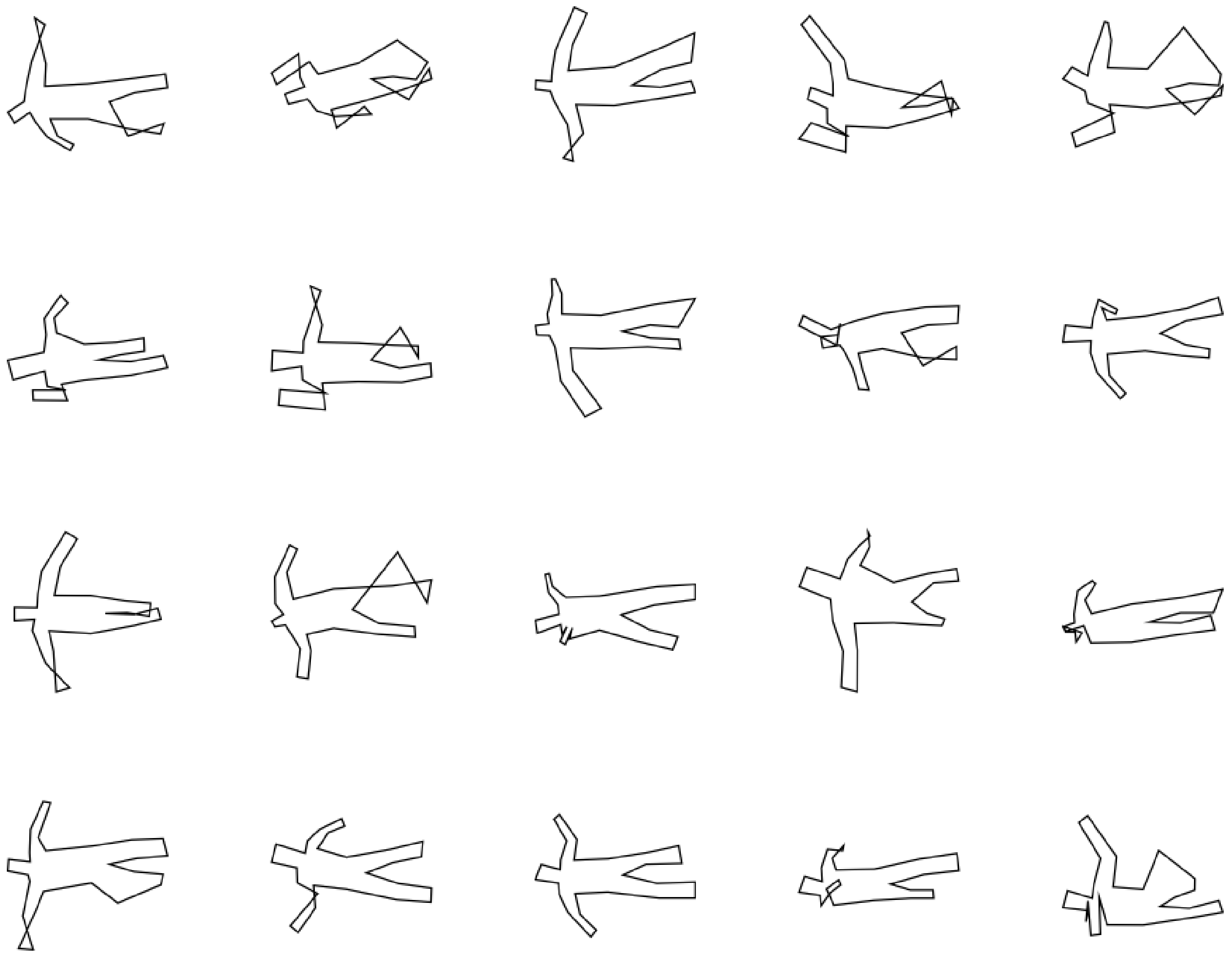


Figure 3.20: Samples from grammar after 30 rounds of EM, using structure 2. From multiple tuning experiment.

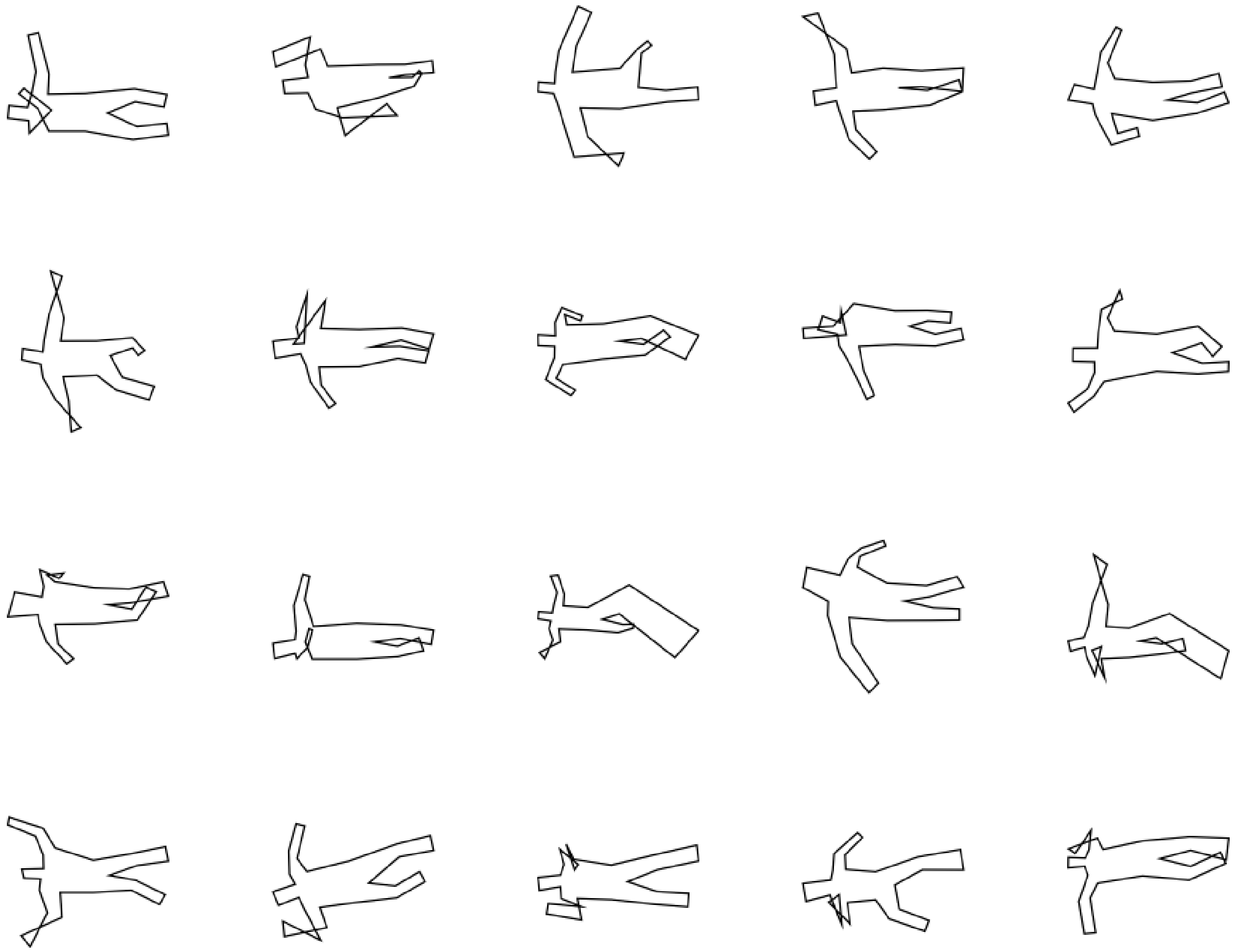


Figure 3.21: Samples from grammar after 30 rounds of EM, using structure 3. From multiple tuning experiment.

3.6 Multiple Midpoints, Multi-Level Correlations

In this experiment, we enrich the grammar by adding in several copies of each nonterminal, each of which has several copies of the original rule with perturbed midpoints. If our original rule was $X \to YZ$, then we have five copies each of X, Y, Z, and each X_i has five rules of the form $X_i \to Y_j Z_k$, where j and k are chosen independently at random. The midpoint distribution for each copy of the rule has its mode randomly perturbed to allow the different copies of a rule to explain different data.

In Figures 3.22, 3.23, 3.24, we show samples from the grammar built using each of the three starting structures. The silhouettes look much better than in the previous experiment. This is because we have learned correlations between levels, i.e., the choice of rule at a higher level influences the choice of rule at a lower level. For example, knowing where we have placed the point at the end of the arm should give us information about where to place the points of the elbow. Ideally, we would have five different variants of the nonterminal corresponding to the arm, each with its own substructure. This results in cleanly articulating parts.

We repeat this experiment three times with different initial grammatical structures, to show how the performance of the algorithm depends on our initial structure. We show the log-likelihoods of the validation data after training in Table 3.1.

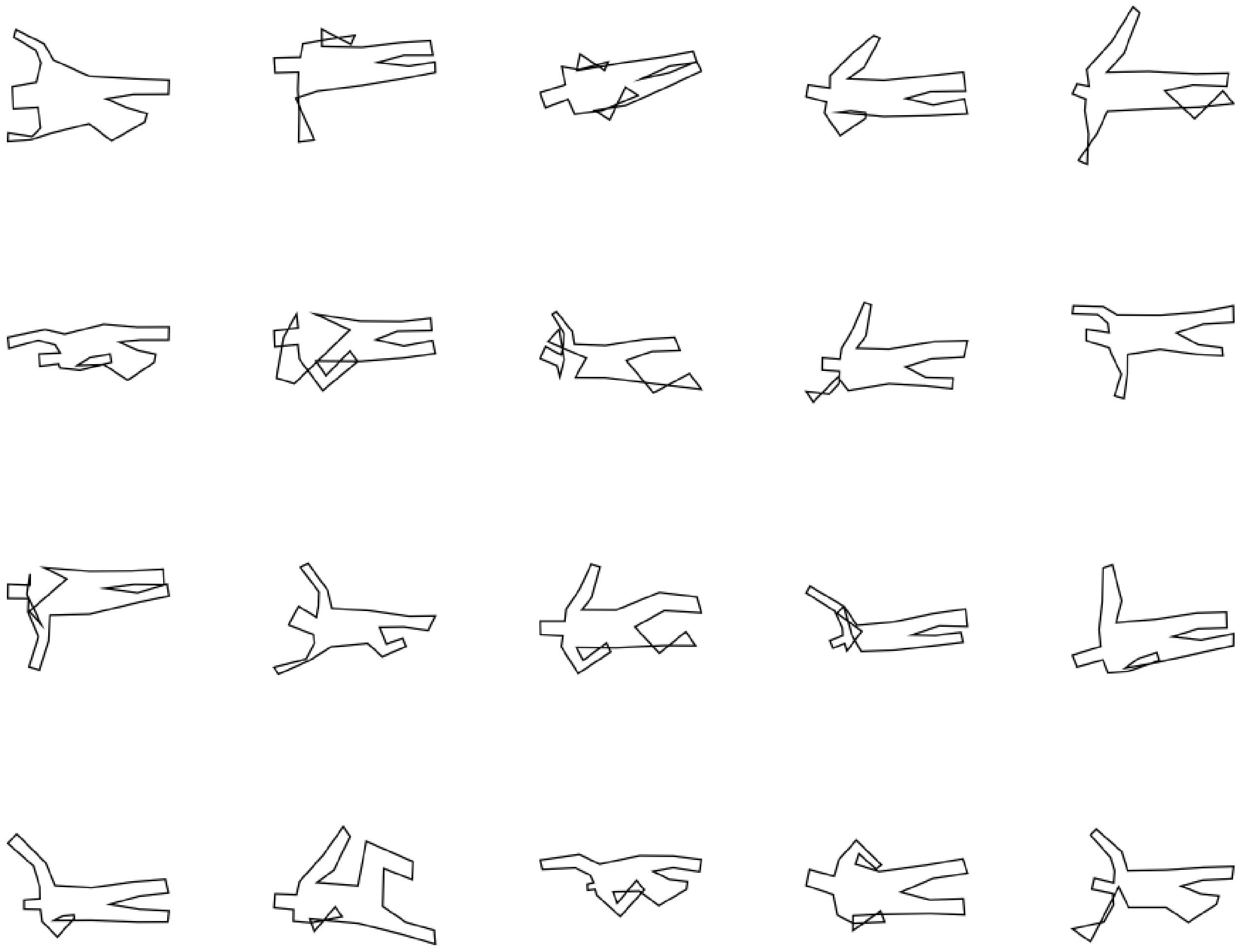


Figure 3.22: Samples from grammar after 30 rounds of EM, using structure 1. From experiment with multi-level correlations.

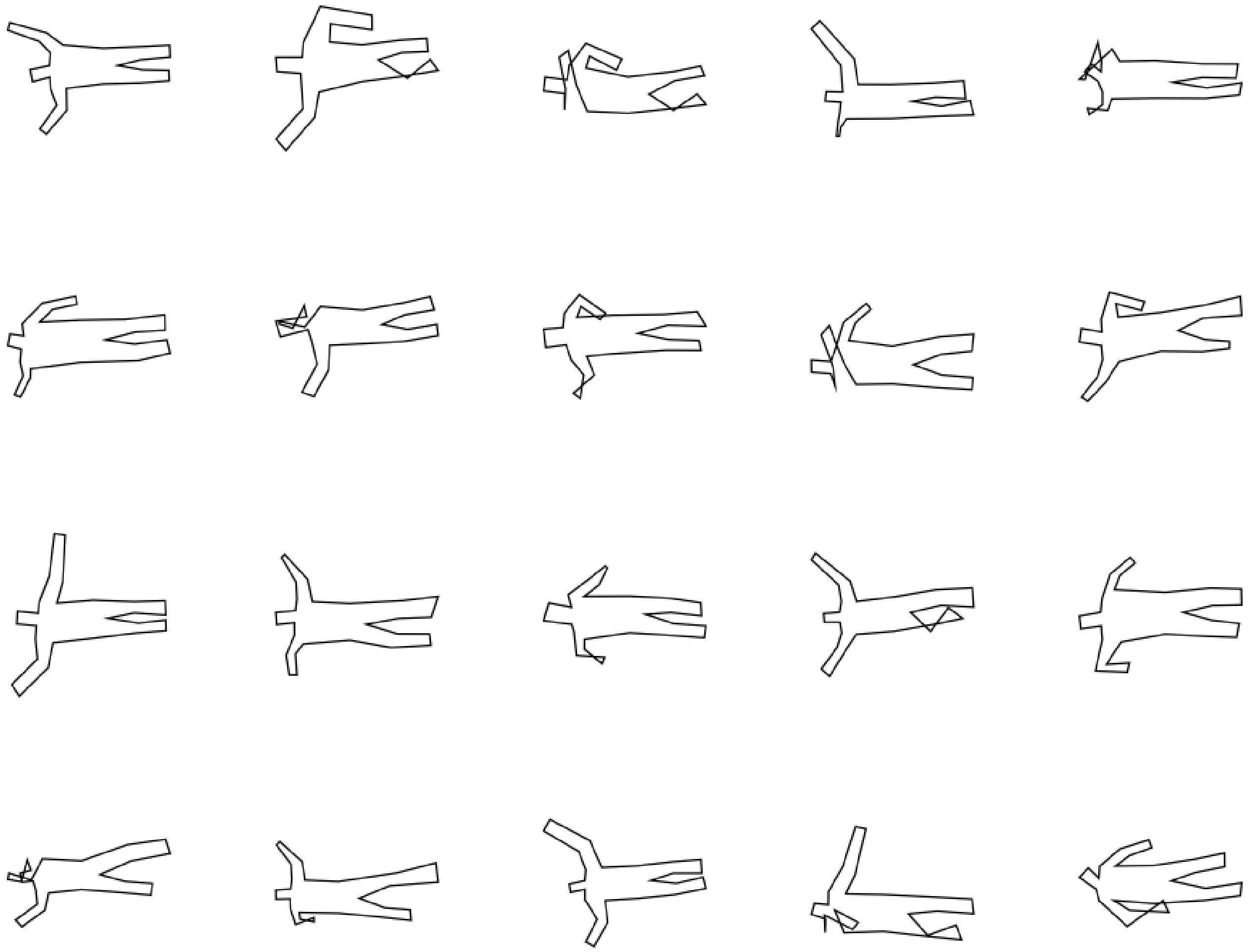


Figure 3.23: Samples from grammar after 30 rounds of EM, using structure 2. From experiment with multi-level correlations.

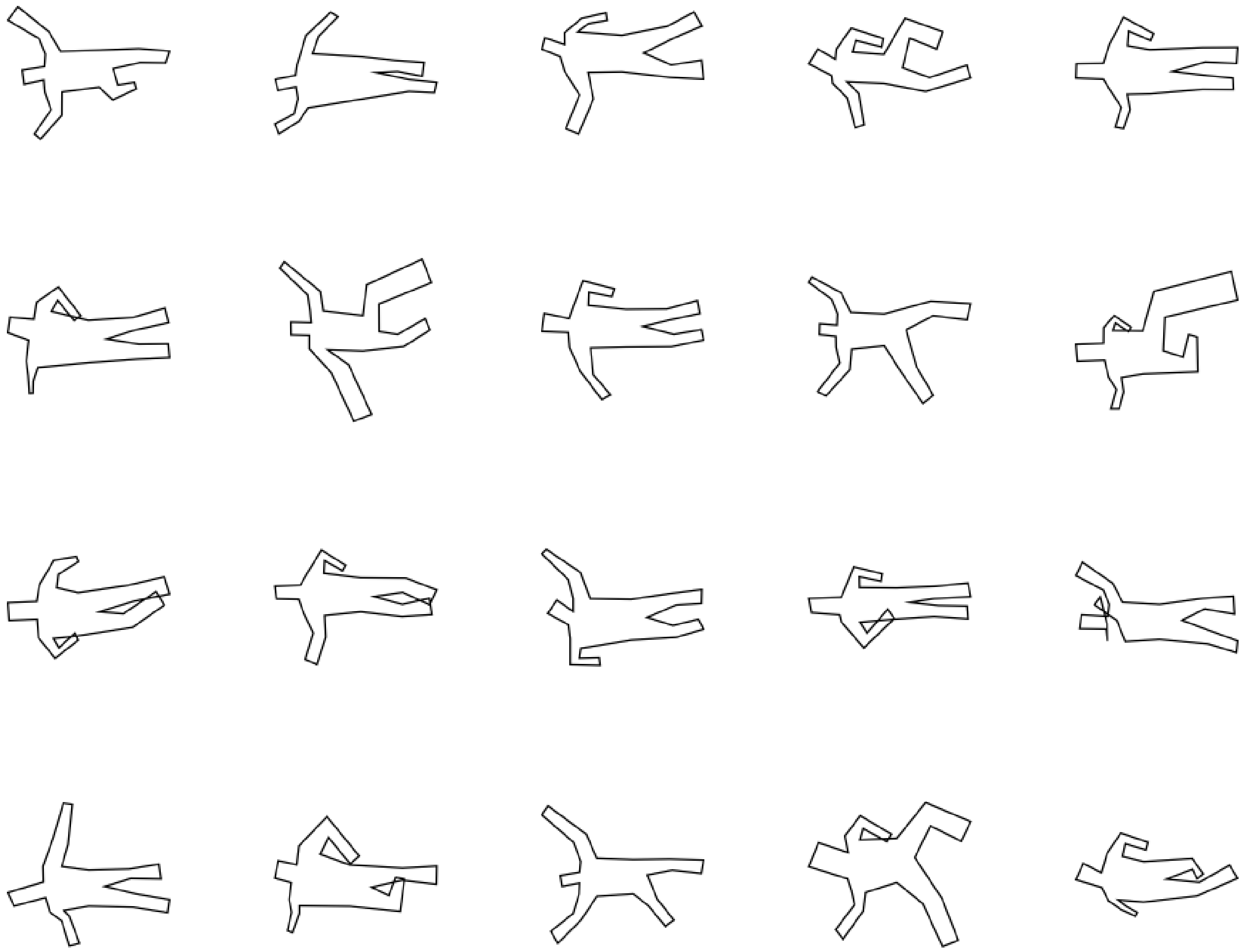


Figure 3.24: Samples from grammar after 30 rounds of EM, using structure 3. From experiment with multi-level correlations.

3.7 Full Tuning

In this experiment, we build a grammar from the example curve that allows all possible decompositions. The grammar has one symbol for every subcurve of the example curve, and one production for every decomposition of a subcurve into two smaller subcurves. We then train using EM. Below we show samples from the grammar after training. Note that there is no longer a choice of initial structure, as we are generating a rule form every possible decomposition of the example curve. We show the log-likelihood of the validation data in Table 3.1.

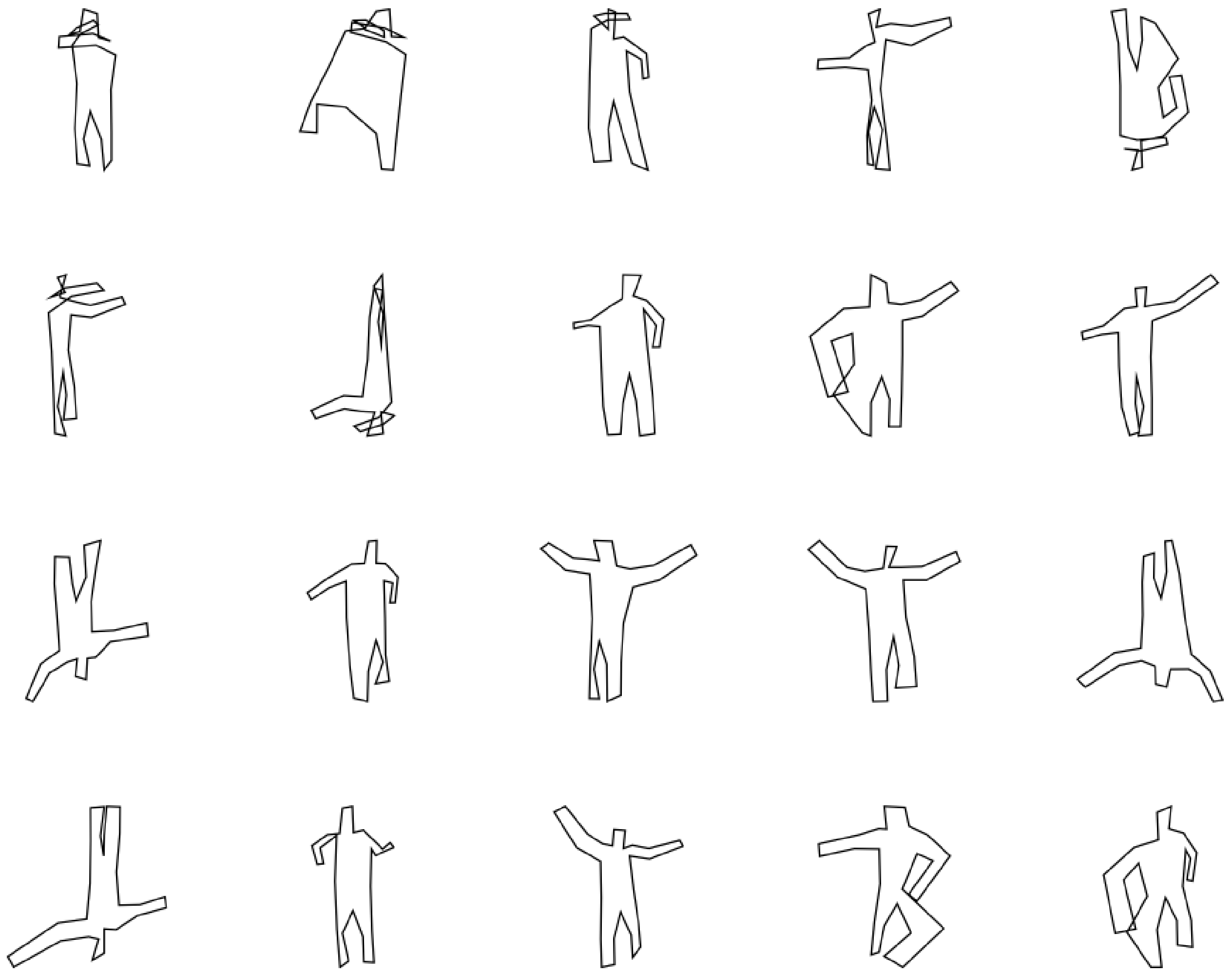


Figure 3.25: Samples from grammar after 30 rounds of EM. From full tuning experiment.

3.8 Comparing to Other Models

3.8.1 Independent Gaussians

In Figures 3.26, 3.27, 3.28, we see samples from a model using an independent Gaussian distribution for each point. They are trained on the same dataset as the grammatical models. Respectively, they show samples from the maximum likelihood model, samples from the maximum likelihood model with no variance, and samples from the maximum likelihood model with its variance artificially decreased. It is plain to see that these models are unable to capture much interesting structure, especially when compared to our models.

We also show (in Figure 3.30) samples from a model of Cootes et al. [1995] trained on hand shapes, which is a model based on non-independent Gaussians.

3.8.2 Comparing to Independent Nonparametric Distributions

In Figure 3.29, we see samples from a model using independent nonparametric distributions for each point. The independence assumption is clearly not a good fit for modeling these curves.

3.9 Classification Experiment

We demonstrate the effectiveness of our models and training procedure by classifying shapes from the Swedish Leaves dataset. For each class, we build a grammar from the first example curve, and train it on all the examples in the training set. Our grammar includes, for each symbol, a shortening rule $X \to \ell$, and for every symbol that is not on the left hand side of any other rule, a lengthening rule $X \to XX$.

The performance achieved is shown in Table 3.2. We also show samples from the models learned in Appendix A.

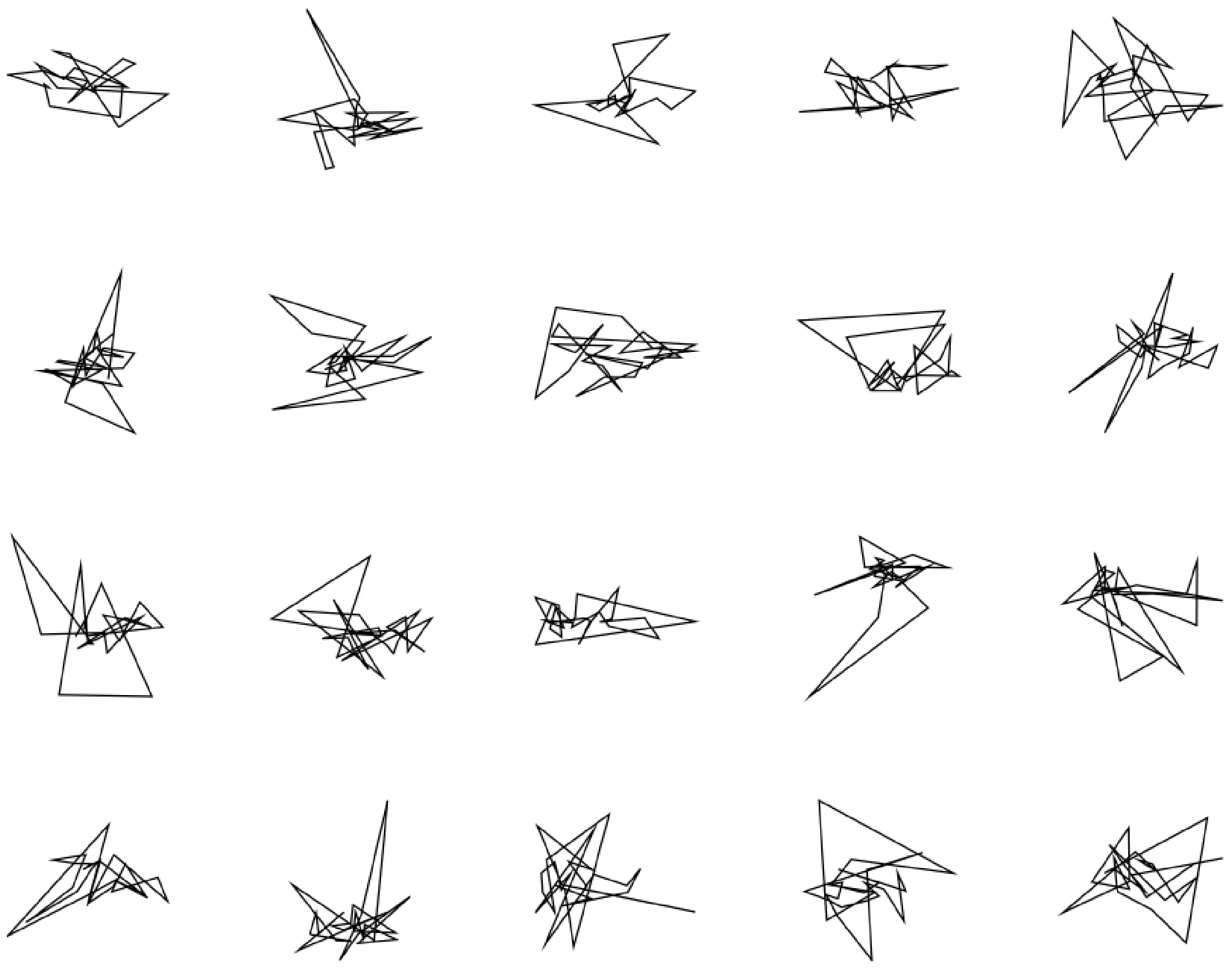


Figure 3.26: Samples from a model using independent Gaussians. We are using the maximum likelihood estimate of parameters.

Method	Recognition Rate
Shape-tree (Felzenszwalb and Schwartz [2007])	96.28%
IDSC + DP (Ling and Jacobs [2007])	94.13%
Fourier descriptors (Ling and Jacobs [2007])	89.60%
Grammatical Models	**88.40%**
SC + DP (Ling and Jacobs [2007])	88.12%
Söderkvist (Söderkvist [2001])	82.40%

Table 3.2: Classification results on the Swedish leaves dataset.

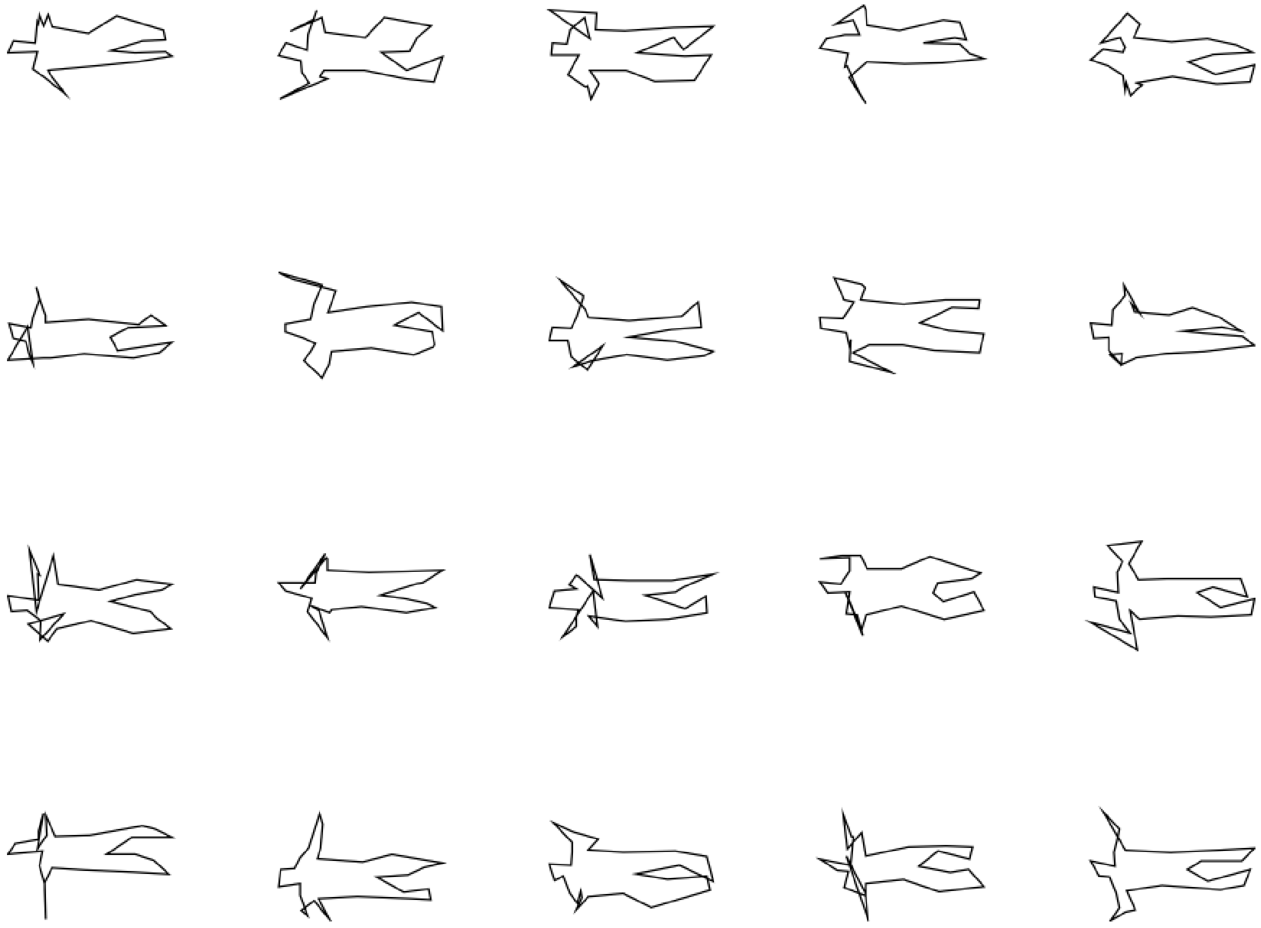


Figure 3.27: Samples from a model using independent Gaussians. The variance of the Gaussians has been decreased to show the means better.

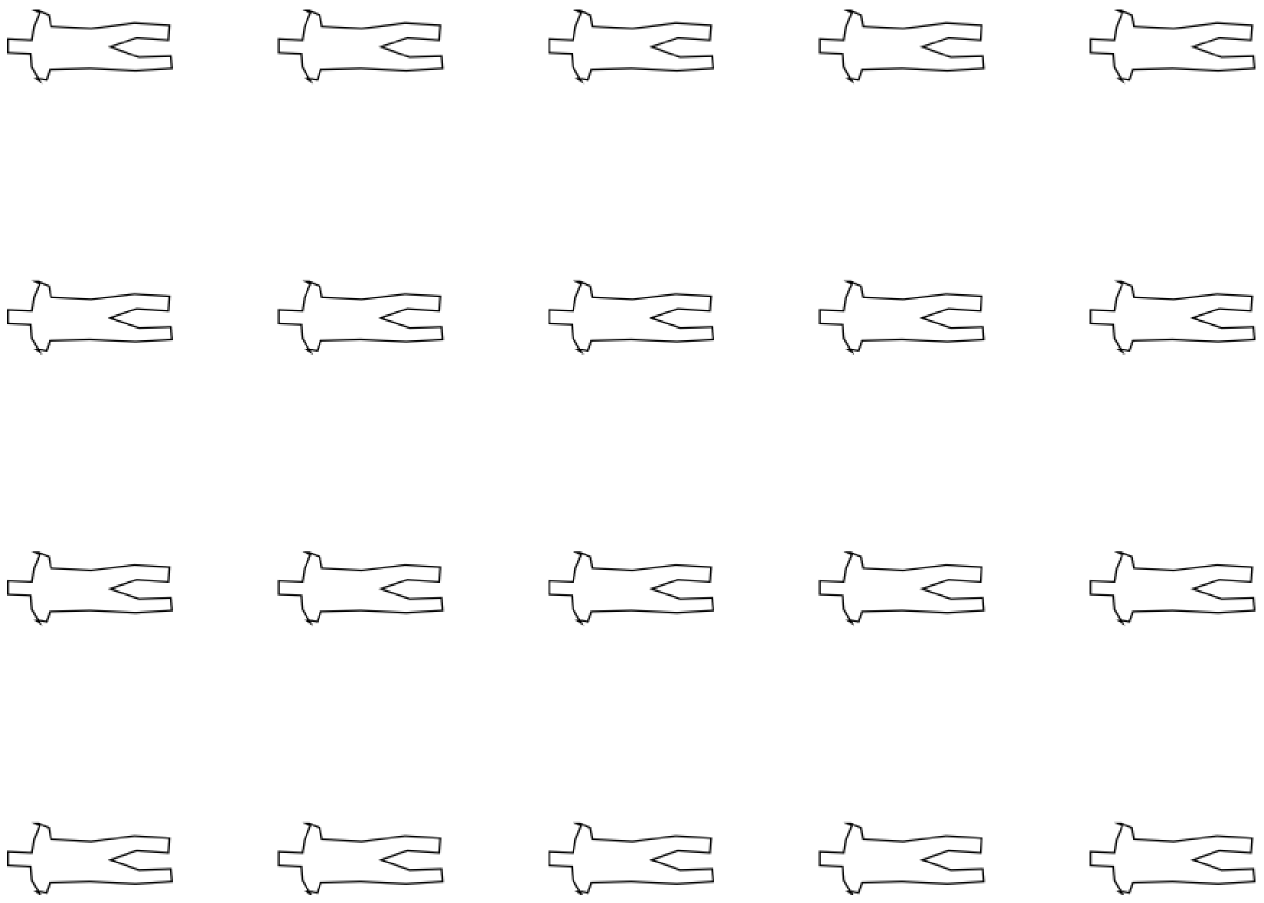


Figure 3.28: Samples from a model using independent Gaussians. The variance of the Gaussians has been set to zero to show the means better.

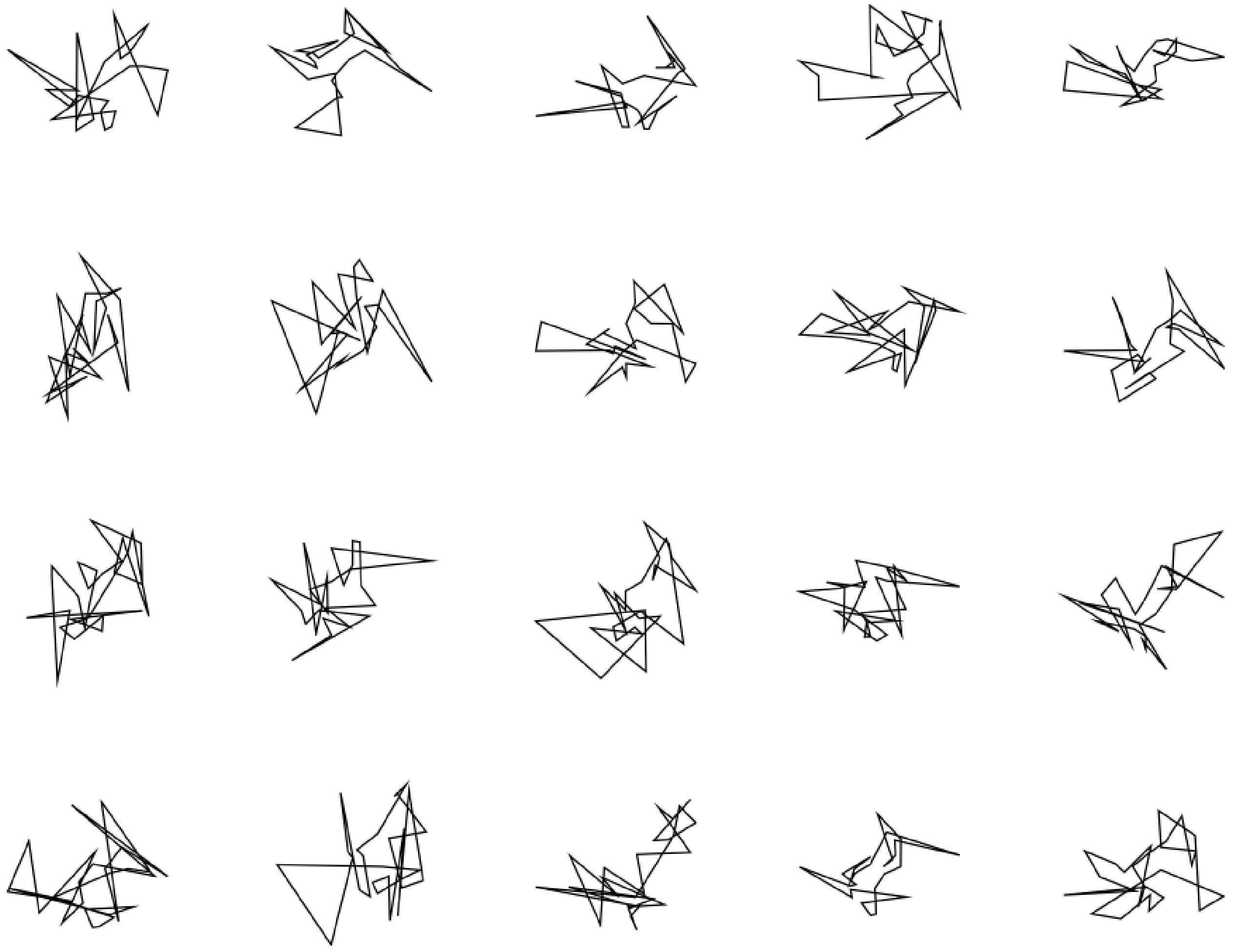


Figure 3.29: Samples from a model using independent nonparametric distributions for each point.

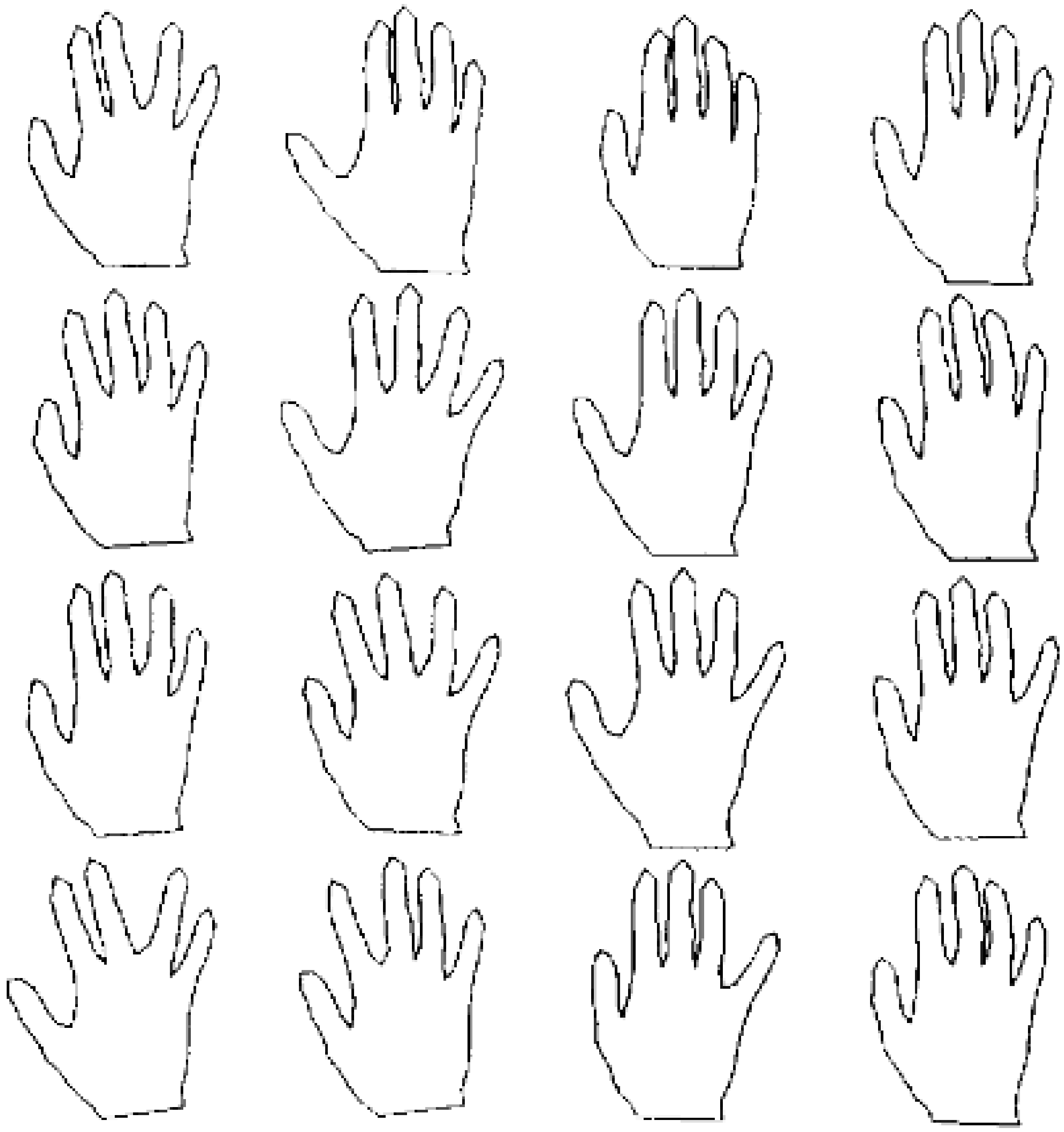


Figure 3.30: Samples from the model of Cootes et al. [1995], from which this figure is taken.

CHAPTER 4
DETECTING OBJECTS IN CLUTTERED IMAGES

Given a shape model and an image containing a similar shape, we would like to be able to find the shape in the image. This is done by searching for a shape that is rated highly by the shape model, and for which evidence exists in the image. Related work has been done in Felzenszwalb and Schwartz [2007], Jin and Geman [2006], Felzenszwalb and McAllester [2010], Felzenszwalb [2005], each concerned with the problem of finding objects in the plane with recursive costs defined by a hierarchical model.

The rest of this chapter is organized as follows: First, we describe filtration parsing, an admissible heuristic method for speeding up a wide array of dynamic programming algorithms, which we have used to speed up parsing. We then describe our formulation of object detection with grammatical models. Finally, we identify unintended reuse as an important obstacle to using grammatical methods for object detection, and describe our approach to solving this problem.

4.1 Speeding up Detection with Filtration Parsing

In this section, we describe filtration parsing, a new algorithm that allows us to speed up certain dynamic programming problems using a technique akin to branch and bound. Similar techniques have been studied before in Felzenszwalb and McAllester [2007] and Raphael [2001].

4.1.1 Lightest Derivation Problems

We now discuss filtration parsing, which is a technique to speed up parsing by eliminating unlikely parses from consideration. It is similar in nature to Felzenszwalb and McAllester [2007], but it can be applied in a more general context. We first require some preliminaries.

We will write a binary tree with root node v, left subtree T_1, and right subtree T_2 as $\frac{v}{T_1 \quad | \quad T_2}$. We will also have non-binary trees, which we will write as

$$\frac{v}{T_1 \quad | \quad \ldots \quad | \quad T_k}.$$

Recall the definition (Definition 2.6.2) of $T_1 \triangle_v^{u} T_2$ from Chapter 2: Let T_1, T_2 be labeled binary trees, where u is a leaf node of T_1 and v is the root node of T_2. We define a new tree $T_1 \triangle_v^{u} T_2$ to be the tree resulting from deleting u and attaching T_2 in its place.

A lightest derivation problem (Felzenszwalb and McAllester [2007]) is a tuple $(\mathcal{S}, \mathcal{R})$, where $\mathcal{S}$ is a set of statements and $\mathcal{R}$ is a set of weighted inference rules. We require that $\mathcal{S}$ contain $\perp$, a special goal-statement. We will denote the weight of a rule $R \in \mathcal{R}$ by $w(R)$.

The goal is to find the lightest (i.e., lowest cost) derivation of the goal statement using the rules in $\mathcal{R}$. Our inference rules are of the following form:

$$\begin{array}{ll} A_1 : w_1 & \\ \vdots & \\ A_k : w_k & \\ \overline{} & v \\ C : v + w_1 + \cdots + w_k. & \end{array}$$

Here the antecedents A_i and the conclusion C are statements in $\mathcal{S}$, v is the cost of using the rule in a derivation, and w_i is the cost of deriving A_i. We will also denote such a rule by $\langle C \Dashv_v A_1, \ldots, A_k \rangle$. We write the conclusion on the left to maintain consistency with the conventional notation for context-free grammars. Note that the antecedents A_i are unordered, so that $\langle C \Dashv_v A_1, A_2 \rangle$ and $\langle C \Dashv_v A_2, A_1 \rangle$ would be considered equivalent.

A derivation of C is a finite tree whose root is labelled with a rule $\langle C \Dashv_v A_1, \ldots, A_k \rangle$, where the subtree rooted at the i-th child is a derivation of A_i. The leaves of this tree are rules with no antecedents. The cost of a tree is the sum of the weights of the rules at its nodes. We will denote the cost of a tree T by $wt(T)$.

Usually, we will work with lightest derivation problems that are in Chomsky Normal Form, i.e., inference rules will either have two antecedents,

$$\frac{\begin{array}{l} A : w_A \\ B : w_B \end{array}}{C : v + w_A + w_B} \; v$$

in which case they will be called *binary rules*, or they will have no antecedents,

$$\frac{}{A : v,} \; v$$

in which case they will be called *lexical rules.*

We will be interested only in lightest derivation problems that are acyclic. A lightest derivation problem $(\mathcal{S}, \mathcal{R})$ is *acyclic* if there is an ordering $\leq$ of the statements in $\mathcal{S}$ such that, for each rule $\langle C =\!|_v\, A_1, \ldots, A_k \rangle \in \mathcal{R}$, $A_i \leq C$ for $i = 1, \ldots, k$.

In most cases, the lightest derivation problem will be specified implicitly by listing abstracts statements and rules which contain free variables. The lightest derivation problem is then obtained by substituting all possible values for each free variable in the abstract statements and rules to create concrete statements and rules. In such cases, listing all the inference rules may be prohibitively slow.

As an example, we can consider CKY parsing with PCFG's as a lightest derivation problem. Suppose our a grammar is in Chomsky Normal Form, and we are parsing a string $s_1, \ldots, s_n$. The lightest derivation problem will have statements $X(i, j)$ for every nonterminal

X and every pair of integers i, j with $1 \le i \le j \le n$. $X(i, j)$ will represent the statement that nonterminal X has the yield $s_i, \dots, s_j$. The rules will be, for every binary rule $X \to YZ$ in the grammar, and all i, j, k with $1 \le i \le j \le k \le n$,

$$\begin{array}{ll} Y(i,j) : w_1 & \\ Z(j+1,k) : w_2 & \\ \rule{2cm}{0.4pt} & w(X \to YZ) \\ X(i,k) : w_1 + w_2 + w(X \to YZ) & \end{array}$$

and, for every nonterminal X, and for every i such that the lexical rule $X \to s_i$ exists,

$$\begin{array}{ll} \rule{2cm}{0.4pt} & w(X \to s_i) \\ X(i,i) : w(X \to s_i), & \end{array}$$

The cost $w(X \to \lambda)$ will be the negative log of the transition probability specified by the PCFG. The goal statement $\bot$ is in this case $S(1, n)$, where S is the start symbol of the grammar.

In this case, listing all the concrete inference rules requires $O(mn^3)$ time, where m is the number of productions in the PCFG. This is the running time of the CKY algorithm, which is too slow to use in many contexts.

4.1.2 Solving Lightest Derivation Problems via Dynamic Programming

Let $\Delta_{\mathcal{S},\mathcal{R}}(C)$ be the set of derivations of C. We define the *cost* of C to be

$$cost_{\mathcal{S},\mathcal{R}}(C) = \min_{T \in \Delta_{\mathcal{S},\mathcal{R}}(C)} wt(T).$$

Algorithm 1 *lightest_derivation*$(\mathcal{S}, \mathcal{R})$

```
Input: $\mathcal{S}, \mathcal{R}$
  $COST \leftarrow \emptyset$
  $BEST \leftarrow \emptyset$
  for $X \in \mathcal{S}$ (in $\leq$ order) do
    $COST[X] \leftarrow \infty$
    for $\langle X \dashv_v \rangle \in \mathcal{R}$ do
      if $COST[X] > v$ then
        $COST[X] \leftarrow v$
        $BEST[X] \leftarrow \langle X \dashv_v \rangle$
      end if
    end for
    for $\langle X \dashv_v YZ \rangle \in \mathcal{R}$ do
      $new \leftarrow COST[Y] + COST[Z] + v$
      if $new < COST[X]$ then
        $COST[X] \leftarrow new$
        $BEST[X] \leftarrow \langle X \dashv_v YZ \rangle$
      end if
    end for
  end for
  return $COST, BEST$
```

Observation 4.1.1. Let $(\mathcal{S}, \mathcal{R})$ be an acyclic[1] lightest derivation problem. We can construct the set $\Delta_{\mathcal{S},\mathcal{R}}(C)$ recursively as follows:

$$\Delta_{\mathcal{S},\mathcal{R}}(C) = \left\{ \frac{\langle C \dashv_v A_1, \ldots, A_k \rangle}{T_1 \quad | \quad \ldots \quad | \quad T_k} \mid \langle C \dashv_v A_1, \ldots, A_k \rangle \in \mathcal{R}, T_i \in \Delta_{\mathcal{S},\mathcal{R}}(A_i) \right\}$$

Observation 4.1.2. Let $(\mathcal{S}, \mathcal{R})$ be an acylic lightest derivation problem. We can recursively compute $cost_{\mathcal{S},\mathcal{R}}(C)$ as

$$cost(C) = \min_{C \dashv_v A_1, \ldots, A_k} v + \sum_i cost(A_i).$$

We can use Observation 4.1.2 in Algorithm 1, which assumes that $(\mathcal{S}, \mathcal{R})$ is an acylic lightest derivation problem in Chomsky Normal Form. Note that this is a standard use of dynamic programming.

1. In the case of a cyclic lightest derivation problem, this definition would also allow infinite trees.

4.1.3 Contexts

In this section, we define "contexts", which are a sort of dual concept to derivations. A B-context for C is a derivation of B that is missing a derivation of C. One node of a B-context for C will be labeled with $hole(C)$ to denote where a derivation of C could be attached. See also Felzenszwalb and McAllester [2007].

Definition 4.1.3. A B-context for C is either the single-node tree $\frac{hole(C)}{\cdot}$ if $B = C$, or a finite tree whose root is labelled with a rule $\langle B =\!\!\mid_v A_1, \dots, A_k \rangle$, where the subtree rooted at the i-th child is either a derivation of A_i or an A_i-context for C, and exactly one of the subtrees is an A_i-context rather than a derivation. The cost of a context T (denoted $wt(T)$) is the sum of the cost of the rules at all of its nodes. Note that the node labelled $hole(C)$ has cost 0.

Let $\Gamma_{\mathcal{S},\mathcal{R}}(C, A)$ be the set of A-contexts for C, and let $\Gamma_{\mathcal{S},\mathcal{R}}(C)$ be the set of $\perp$-contexts of C. Define

$$ocost_{\mathcal{S},\mathcal{R}}(C) = \min_{T \in \Gamma_{\mathcal{S},\mathcal{R}}(C)} wt(T).$$

ocost is analogous to the outside cost in the Inside-Outside Algorithm for PCFG's, but here we are trying to find a lightest derivation rather than sum over all derivations.

Observation 4.1.4. Let $(\mathcal{S}, \mathcal{R})$ be an acylic lightest derivation problem. We can construct the set $\Gamma_{\mathcal{S},\mathcal{R}}(C)$ recursively as follows:

- $\Gamma_{\mathcal{S},\mathcal{R}}(\perp) = \left\{ \frac{hole(\perp)}{\cdot} \right\}$. (Note that, since the lightest derivation problem is assumed to be acyclic, this is the only $\perp$-context for $\perp$.)

- $$\Gamma_{\mathcal{S},\mathcal{R}}(X) = \{ \frac{\langle X =\!\!\mid_v A_1, \dots, A_k \rangle}{T_1 \quad | \quad \dots \quad | \quad T_k} \mid$$
$$\text{exactly one } T_i \in \Gamma_{\mathcal{S},\mathcal{R}}(C, A_i), \text{ other } T_i \in \Delta_{\mathcal{S},\mathcal{R}}(A_i) \}$$

Observation 4.1.5. Let $(\mathcal{S}, \mathcal{R})$ be an acyclic lightest derivation problem. Given $cost(X)$ for all X, we can recursively compute $ocost(X)$:

$$\begin{aligned} ocost(\perp) &= 0 \\ ocost(C) &= \min_{A \dashv_v C, B_1, \ldots, B_{k-1} \in \mathcal{R}} ocost(A) + v + \sum_i cost(B_i) \end{aligned}$$

This is implemented by Algorithm 2, which assumes that our lightest derivation problem is in Chomsky Normal Form.

Algorithm 2 *outside*$(\mathcal{S}, \mathcal{R}, COST)$

```
Input: $\mathcal{S}, \mathcal{R}, COST$
  $OCOST \leftarrow \emptyset$
  for $X \in \mathcal{S}$ do
    $OCOST[X] \leftarrow \infty$
  end for
  $OCOST[\perp] \leftarrow 0$
  for $X \in \mathcal{S}$ (in reverse $\leq$ order) do
    for $X \dashv_v YZ \in \mathcal{R}$ do
      $OCOST[Y] \leftarrow \min\{OCOST[Y], v + OCOST[X] + COST[Z]\}$
      $OCOST[Z] \leftarrow \min\{OCOST[Z], v + OCOST[X] + COST[Y]\}$
    end for
  end for
  return $OCOST$
```

4.1.4 Homomorphisms

Definition 4.1.6 (Homomorphism)**.** Let $(\mathcal{S}, \mathcal{R})$ and $(\mathcal{S}', \mathcal{R}')$ be lightest derivation problems. A *homomorphism* of lightest derivation problems is a function $\Phi : \mathcal{S} \to \mathcal{S}'$ such that, for every $\langle C \dashv_v A_1, \ldots, A_k \rangle \in \mathcal{R}$, there exists a rule $\langle \Phi(C) \dashv_w \Phi(A_1), \ldots, \Phi(A_k) \rangle \in \mathcal{R}'$, with $w \leq v$. We also require that $\Phi(\perp) = \perp$.

Homomorphisms are called "abstractions" in Felzenszwalb and McAllester [2007]. We define the specific homomorphisms used in Section 4.2.1.

Lemma 4.1.7. *Let Φ be a homomorphism of lightest derivation problems. Then $cost_{\mathcal{S},\mathcal{R}}(X) \geq cost_{\mathcal{S}',\mathcal{R}'}(\Phi(X))$. Also, $ocost_{\mathcal{S},\mathcal{R}}(X) \geq ocost_{\mathcal{S}',\mathcal{R}'}(\Phi(X))$.*

Proof. Let T be the derivation achieving $cost(\mathcal{S},\mathcal{R})(X)$. We can replace each rule $\langle C =\!\mid_v A_1, \ldots, A_k \rangle$ used in the derivation with a rule $\langle \Phi(X) =\!\mid_w \Phi(A_1), \ldots \Phi(A_k) \rangle$, thereby creating a valid derivation of $\Phi(X)$ whose total cost is lower than that of T. Thus the lightest derivation of $\Phi(X)$ also has cost lower than that of T.

A similar argument holds for contexts and $ocost_{\mathcal{S},\mathcal{R}}$. □

Definition 4.1.8. Let $(\mathcal{S},\mathcal{R})$ be a lightest derivation problem. A table C mapping statements to weights is a *valid lower bound* for $(\mathcal{S},\mathcal{R})$ if, for every $X \in \mathcal{S}$, $cost_{(\mathcal{S},\mathcal{R})}(X) \geq C[X]$.

A table C mapping statements to weights is a *valid lower bound on contexts* for $(\mathcal{S},\mathcal{R})$ if, for every $X \in \mathcal{S}$, $ocost_{(\mathcal{S},\mathcal{R})}(X) \geq C[X]$.

Lemma 4.1.9. *If $\Phi : \mathcal{S} \to \mathcal{S}'$ is a homomorphism, and C' is a valid lower bound for $(\mathcal{S}',\mathcal{R}')$, and we define $C[X] := C'[\Phi(X)]$, then C is a valid lower bound for $(\mathcal{S},\mathcal{R})$. C is called the* pullback *of C' under Φ. The same is true of valid lower bounds on contexts.*

Proof. Let $X =\!\mid A_1, \ldots, A_k$ be the rule that achieves X's lightest derivation. Then

$$cost_{\mathcal{S},\mathcal{R}}(X) \geq cost_{\mathcal{S}',\mathcal{R}'}(\Phi(X))$$

by Lemma 4.1.7

$$\geq C'[\Phi(X)]$$

because C' is a valid lower bound for $\mathcal{S}', \mathcal{R}'$

$$= C[X]$$

The proof for lower bounds on contexts is analogous. $\square$

Observation 4.1.10. Every derivation of $\perp$ in which X appears can be realized as

$$T_1 \triangle^{\cancel{hole(X)}} T_2,$$

where $T_1 \in \Gamma_X$ and $T_2 \in \Delta_X$. Therefore, if $cost(X) = \min_{T \in \Delta_X} wt(T)$ and $ocost(X) = \min_{T \in \Gamma_X}$, then the lightest derivation of $\perp$ using X has cost $cost(X) + ocost(X)$.

Lemma 4.1.11. *If T is a lightest derivation of $\perp$, and X does not appear in T, then the value of $\mathcal{S}, \mathcal{R}$ is the same as the value of $\mathcal{S} \setminus \{X\}, \mathcal{R}'$, where $\mathcal{R}'$ is $\mathcal{R}$ minus any rules involving X.*

4.1.5 Putting it All Together

Theorem 4.1.12. *Let $(\mathcal{S}, \mathcal{R})$ be a lightest derivation problem, and let X be a statement. Suppose that $\underline{COST}$ is a valid lower bound for $(\mathcal{S}, \mathcal{R})$, and $\underline{OCOST}$ is a valid lower bound on contexts for $(\mathcal{S}, \mathcal{R})$. Then the lightest derivation that uses X has cost at least $\underline{COST}[X] + \underline{OCOST}[X]$.*

Consequently, if there is a derivation T of $\perp$ with cost $wt(T) < \underline{COST}[X] + \underline{OCOST}[X]$, then no lightest derivation of $\perp$ uses X.

We can use this theorem to perform an admissible coarse-to-fine search strategy. Consider a sequence of LDP's $(\mathcal{S}_1, \mathcal{R}_1), \ldots, (\mathcal{S}_k, \mathcal{R}_k)$, with homomorphisms $\Phi_i : \mathcal{S}_i \to \mathcal{S}_{i+1}$ for $i = 1, \ldots, k-1$. We can solve the problems in reverse order, using the bound of Theorem 4.1.12 to eliminate statements. This is implemented by Algorithms 3, 4, 5, and 6. Algorithms 3 and 4 are very similar to Algorithms 1 and 2, but they do not consider solutions that violate the lower bound of Theorem 4.1.12. Algorithm 6 shows how to use the inside pass and the outside pass to organize our search.

We now give an informal description of the algorithm. We start by solving the coarsest problem $(\mathcal{S}_k, \mathcal{R}_k)$. We then (using the function $lift$) lift the solution found from the coarsest level to the finest level. This is a more standard use of coarse-to-fine search. This gives us a derivation of $\perp$ in $(\mathcal{S}_1, \mathcal{R}_1)$, and the cost of that derivation is an upper bound on the cost of the optimal derivation. Throughout the running of the algorithm, the variable u will be the lowest cost seen for a derivation of $\perp$ in $(\mathcal{S}_1, \mathcal{R}_1)$.

Because of Theorem 4.1.12, we can delete any statement from $(\mathcal{S}_i, \mathcal{R}_i)$ if we can prove that it does not participate in any derivation of cost u or lower. This is not only guaranteed not to change the minimum cost of a derivation in $(\mathcal{S}_i, \mathcal{R}_i)$, it is guaranteed not to delete any statement X' such that

$$X' = \Phi_{i-1}(\Phi_{i-2}(\dots(\Phi_1(X)\dots),$$

for any statement X participating in an optimal derivation of $\perp$ in $(\mathcal{S}_1, \mathcal{R}_1)$. Thus, whenever we delete a statement X' in $(\mathcal{S}_i, \mathcal{R}_i)$, we can also delete all statements in finer problems that map to X'.

We alternate between an inside pass, where we compute a valid lower bound for $(\mathcal{S}_i, \mathcal{R}_i)$, and an outside pass, where we compute a valid lower bound on contexts for $(\mathcal{S}_{i-1}, \mathcal{R}_{i-1})$. These are implemented by Algorithms 3 and 4, respectively. Algorithm 3 takes a lower bound on contexts as input, and uses this to delete any statements that do not pass the test of Theorem 4.1.12. This can be done because, in line 12 of Algorithm 3, when we consider X, we have already considered all rules with X on the left-hand side, and thus $\underline{COST}[X]$ is already a lower bound on the cost of any derivation of X that does not use any deleted statements.

Similarly, Algorithm 4 takes a lower bound for $(\mathcal{S}_{i-1}, \mathcal{R}_{i-1})$ (derived in Algorithm 6 by taking the pullback of a valid lower bound for $(\mathcal{S}_i, \mathcal{R}_i)$) as input, and uses this to delete any statements that do not pass the test of Theorem 4.1.12. This can be done because, in line 3 of Algorithm 4, when we consider X, we have already considered all rules that use X on the right-hand side, and thus $\underline{OCOST}[X]$ is at that point a lower bound on the cost of a context for X that does not use any deleted statements.

We prove the correctness of Algorithm 6 with two lemmas:

Lemma 4.1.13. *If $\underline{OCOST}$ is a valid lower bound on contexts for $(\mathcal{S}_i, \mathcal{R}_i)$, and u is an upper bound on the optimal cost of a derivation of $\perp$ in $(\mathcal{S}_i, \mathcal{R}_i)$, then after*

$$\underline{COST}, T \leftarrow inside(u, (\mathcal{S}_i, \mathcal{R}_i), \underline{OCOST}),$$

$\underline{COST}$ is a valid lower bound for $(\mathcal{S}_i, \mathcal{R}_i)$. Note that inside deletes some statements and rules from $(\mathcal{S}_i, \mathcal{R}_i)$.

Lemma 4.1.14. *If $\underline{COST}$ is a valid lower bound for $(\mathcal{S}_{i-1}, \mathcal{R}_{i-1})$, and u is an upper bound on the optimal cost of a derivation of $\perp$ in $(\mathcal{S}_{i-1}, \mathcal{R}_{i-1})$, then after*

$$\underline{OCOST} \leftarrow outside(u, \mathcal{S}_{i-1}, \mathcal{R}_{i-1}, \underline{COST}),$$

$\underline{OCOST}$ is a valid lower bound on contexts for $(\mathcal{S}_{i-1}, \mathcal{R}_{i-1})$. Note that outside deletes some statements and rules from $(\mathcal{S}_{i-1}, \mathcal{R}_{i-1})$.

4.1.6 Inadmissible Homomorphisms

We can apply this algorithm even when our homomorphisms do not satisfy the required lower bounds. In this case, we are not guaranteed to find an optimal solution. Designing inadmissible homomorphisms that yield good results in practice is an important area of future work.

Algorithm 3 $\mathit{inside}(\mathcal{S}, \mathcal{R}, \underline{OCOST}, u)$

```
Input: $\mathcal{S}, \mathcal{R}, \underline{OCOST}, u$
 1: $\underline{COST} \leftarrow \emptyset$
 2: $BEST \leftarrow \emptyset$
 3: for $X \in \mathcal{S}$ do
 4:   $\underline{COST}[X] \leftarrow \infty$
 5:   for $\langle X \dashv_v \rangle \in \mathcal{R}$ do
 6:     if $\underline{COST}[X] > v$ then
 7:       $\underline{COST}[X] \leftarrow v$
 8:       $BEST[X] \leftarrow X \dashv_v$
 9:     end if
10:   end for
11: end for
12: for $X \in \mathcal{S}$ (in $\leq$ order) do
13:   if $\underline{COST}[X] + \underline{OCOST}[X] > u$ then
14:     delete $X$ from $\mathcal{S}$
15:     continue
16:   end if
17:   for $Z \dashv_v XY \in \mathcal{R}$ do
18:     $\mathit{new} \leftarrow \underline{COST}[X] + \underline{COST}[Y] + v$
19:     if $\mathit{new} < \underline{COST}[Z]$ then
20:       $\underline{COST}[Z] \leftarrow \mathit{new}$
21:       $BEST[Z] \leftarrow Z \dashv_v XY$
22:     end if
23:   end for
24: end for
25: return $\underline{COST}, BEST$
```

Algorithm 4 $\mathit{outside}(\mathcal{S}, \mathcal{R}, \underline{COST}, u)$

```
Input: $\mathcal{S}, \mathcal{R}, \underline{COST}, u$
 1: $\underline{OCOST} \leftarrow \emptyset$
 2: $\underline{OCOST}[\perp] \leftarrow 0$
 3: for $X \in \mathcal{S}$ (in reverse $\leq$ order) do
 4:   if $\underline{COST}[X] + \underline{OCOST}[X] > u$ then
 5:     delete $X$ from $\mathcal{S}$
 6:     continue
 7:   end if
 8:   for $X \dashv_v YZ \in \mathcal{R}$ do
 9:     $\underline{OCOST}[Y] \leftarrow \min\{\underline{OCOST}[Y], v + \underline{OCOST}[X] + \underline{COST}[Z]\}$
10:     $\underline{OCOST}[Z] \leftarrow \min\{\underline{OCOST}[Z], v + \underline{OCOST}[X] + \underline{COST}[Y]\}$
11:   end for
12: end for
13: return $\underline{OCOST}$
```

Algorithm 5 $lift(\{(\mathcal{S}_i, \mathcal{R}_i)\}_{i=1}^{\ell}, \{\Phi_i\}_{i=1}^{\ell-1}, T)$

```
Input: $\{(\mathcal{S}_i, \mathcal{R}_i)\}_{i=1}^{\ell}, \{\Phi_i\}_{i=1}^{\ell-1}, T$
  $T_\ell \leftarrow T$
  for $i = \ell - 1$ downto 1 do
    for $X \in \mathcal{S}_i$ such that $\Phi_i(X)$ is used in $T_{i+1}$, in $\leq$ order do
      $COST[X] \leftarrow \infty$
      for $\langle X \dashv_v \rangle \in \mathcal{R}_i$ do
        if $v < COST[X]$ then
          $COST[X] \leftarrow v$
          $BEST[X] \leftarrow \langle X \dashv_v \rangle$
        end if
      end for
      for $\langle X \dashv_v YZ \rangle \in \mathcal{R}_i$ do
        $new \leftarrow v + COST[Y] + COST[Z]$
        if $new < COST[X]$ then
          $COST[X] \leftarrow new$
          $BEST[X] \leftarrow \langle X \dashv_v YZ \rangle$
        end if
      end for
    end for
    Construct $T_i$ by following $BEST[\cdot]$ values starting with $BEST[\perp]$ as root
  end for
  return $T_1, cost(T_1)$
```

Algorithm 6 $solve(\{(\mathcal{S}_i, \mathcal{R}_i)\}, \{\Phi_i\})$

```
Input: $\{(\mathcal{S}_i, \mathcal{R}_i)\}, \{\Phi_i\}$
  $OCOST_k[*] \leftarrow -\infty$
  $u \leftarrow \infty$
  $best \leftarrow \emptyset$
  for $i = k$ downto 1 do
    $\underline{COST}_i, T \leftarrow inside(\mathcal{S}_i, \mathcal{R}_i, \underline{OCOST}_i, u)$
    $cost, soln \leftarrow lift(\{(\mathcal{S}_j, \mathcal{R}_j)\}_{j=1}^{i}, \{\Phi_j\}_{j=1}^{i-1}, T)$
    if $cost < u$ then
      $u \leftarrow cost$
      $best \leftarrow soln$
    end if
    $\underline{OCOST} \leftarrow outside(\mathcal{S}_{i-1}, \mathcal{R}_{i-1}, pullback(\underline{COST}_i, \Phi_{i-1}), u)$
  end for
```

4.2 Parsing in the Plane with Grammatical Models

4.2.1 Plane Parsing

For the problem of detecting an object with a grammar in a cluttered image, we have the following lightest derivation problem: for every pair of points p, q, and every nonterminal X, we have the statement $X(p, q)$.

For every rule $X \to YZ$ in our grammar, and every triple of points p, q, r, we have

$$\begin{array}{ll} Y(p,q) : w_1 & \\ Z(q,r) : w_2 & \\ \hline & v = -\log\left(\mu_{X\to YZ}(p,q,r)\cdot\rho(X\to YZ)\right) \\ X(p,r) : w_1 + w_2 + v & \end{array}$$

and, for every rule $X \to \ell$, and every pair p, q

$$\begin{array}{ll} \hline & v = data(p,q) - \log\rho(X\to\ell) \\ X(p,q) : v & \end{array}$$

where $data(p, q)$ is a data cost modeling the log-likelihood of a line segment existing between p and q. Specifically, $data(p, q)$ is chosen so that the sum over all terms will be $\log\frac{P(I|\text{parse})}{P(I|\text{no object})}$, where I is the image data and the probabilities refer to a probabilistic model of image formation. Our image formation model is described in the next section.

Note that this lightest derivation problem is only acyclic if the grammar used is acyclic. If the grammar used has cycles, we can require that some rules $X(p,r) =\!\!|_v\ Y(p,q), Z(q,r)$ are restricted to p, q, r such that the distance between p and r is at least as big as that between p and q, and q and r.

We construct homomorphisms for filtration parsing by coarsening the image grid, which we represent by pairs of integer indices. Let $\varphi((i,j)) = \left(\left\lfloor \frac{i}{2} \right\rfloor, \left\lfloor \frac{j}{2} \right\rfloor\right)$, and let $\Phi(X(p,q)) = X(\varphi(p), \varphi(q))$. By repeatedly applying Φ, we construct a hierarchy of lightest derivation problems, in which each level has 4 times fewer points, 16 times fewer statements and (roughly) 64 times fewer rules than the previous level.

To make the Φ_i be legitimate homomorphisms, for each binary rule $\langle X(p,r) =\!\!|_v\, Y(p,q), Z(q,r)\rangle$ in $\mathcal{R}_i$, we need a lower bound on the cost of rules mapping to it. We can think of this as finding

$$\min_{p' \in P, q' \in Q, r' \in R} \mu_{X \to YZ}(p', q', r'),$$

where P, Q, R are the sets of grid points in our original lightest derivation problem that map to p, q, r in the coarsened problem.

We also need a lower bound on the cost of rules mapping to lexical rules $\langle X(p,q) =\!\!|_v \rangle$. We do this by simply lifting all costs from the finest level.

4.2.2 Data Models for Object Detection

We use a very straightforward data model, that of a Bernoulli model over Canny edges.

We set $P(E_p = 1 \mid p \in O) = p_{fg}$, $P(E_p = 1 \mid p \notin O) = p_{bg}$.

We use ratio of foreground to background, so that we only have to multiply over pixels in the object.

$$\begin{aligned}
P(E \mid O) &= \prod_{p \in O} p_{fg}^{E_p}(1-p_{fg})^{1-E_p} \cdot \prod_{p \notin O} p_{bg}^{E_p}(1-p_{bg})^{1-E_p} \\
&= \prod_{p \in O} \left(\frac{p_{fg}}{p_{bg}}\right)^{E_p} \left(\frac{1-p_{fg}}{1-p_{bg}}\right)^{1-E_p} \cdot \prod_{p} p_{bg}^{E_p}(1-p_{bg})^{1-E_p} \\
&\propto \prod_{p \in O} \left(\frac{p_{fg}}{p_{bg}}\right)^{E_p} \left(\frac{1-p_{fg}}{1-p_{bg}}\right)^{1-E_p} \\
\log P(E \mid O) &= \sum_{p \in O} \left[E_p \log\left(\frac{p_{fg}}{p_{bg}}\right) + (1-E_p)\log\left(\frac{1-p_{fg}}{1-p_{bg}}\right)\right] + C \\
&= \sum_{p \in O} \left[E_p \log\left(\frac{p_{fg}(1-p_{bg})}{p_{bg}(1-p_{fg})}\right) + \log\left(\frac{1-p_{fg}}{1-p_{bg}}\right)\right] + C \\
&= \sum_{p \in O} \left[E_p \log\left(\frac{p_{fg}(1-p_{bg})}{p_{bg}(1-p_{fg})}\right)\right] + |O| \log\left(\frac{1-p_{fg}}{1-p_{bg}}\right) + C
\end{aligned}$$

We therefore use the data cost

$$data(p,q) = \sum_{r \in \ell_{p,q}} E_r \log\left(\frac{p_{fg}}{p_{bg}}\right) + (1-E_r)\log\left(\frac{1-p_{fg}}{1-p_{bg}}\right).$$

This correctly implements the data cost when there is no reuse of the same pixel by different line segments. We discuss this in more detail in the next section.

When doing scene parsing, we will modify this formula slightly. (This idea was first introduced in Amit and Trouvé [2007].) Pixels that are in already selected objects will be labelled foreground, and other pixels will initially be labelled background. Let F be the set of pixels already marked foreground.

$$\log P(E \mid O) = \sum_{p \in O \setminus F} \left[E_p \log\left(\frac{p_{fg}(1-p_{bg})}{p_{bg}(1-p_{fg})}\right)\right] + |O \setminus F| \log\left(\frac{1-p_{fg}}{1-p_{bg}}\right) + C$$

For pixels in $F \cap O$, the ratio is one, and thus its log is zero. For pixels not in O, the ratio becomes a constant as before.

4.3 Unintended Reuse and Local Inference

In this section, we discuss local inference, a method we have used to clean up parses that exhibit unintended reuse. Our method is as follows: we first parse the image to find the highest-scoring curve, with or without unintended reuse. We then perform our local inference procedure, which we formulate as an energy minimization problem. We then reparse the image, using some constraints derived from the output of the local inference procedure.

Unintended reuse is a serious and fundamental problem in the interpretation of images. Here, we define the problem and argue that it is not possible to avoid this problem using standard context-free grammatical models.

We can consider an interpretation of an image to consist of various assertions about the contents of the image. We can characterize these assertions by asking what region of the image they refer to, and in particular how large that region is.

Compositional models are those models in which the quality or plausibility of a high-level assertion (i.e., one that refers to a large image region) is defined recursively in terms of the quality of lower-level assertions. In order to make these problems amenable to dynamic programming, we must assume context-freeness. It is thus a general problem of compositional models that we can only efficiently and exactly compute the quality of high-level assertions by assuming context-freeness and allowing them to depend on low-level assertions that are contradictory. This can then result in incoherent interpretations of the image. In our case, straight-forward maximization of the likelihood under the statistical model described above yields curves that have many self intersections; curves in which long contours are used multiple times, often in opposite directions; and curves which do not enclose any area. A parse with unintended reuse is shown in Figure 4.1.

In general, we cannot sacrifice efficiency beyond a certain point, and so we must search

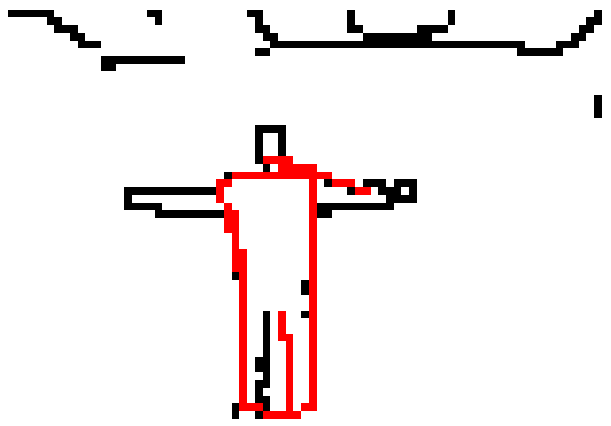


Figure 4.1: A parse exhibiting unintended reuse.

for a coherent interpretation which may not score the highest, even among coherent interpretations.

One approach to solving this problem, used in Jin and Geman [2006], is to fix various low-level assertions that seem likely, and eliminate incompatible low-level assertions from consideration. In our case, this approach does not yield good results.

4.4 State space

We assume that there is a single object of interest against a background, and that its boundary is the curve we wish to detect. Let P be the set of pixels in the input image, and let $E \subset P$ be the pixels marked as edges by some edge detector, such as the Canny edge detector (Canny [1986]). We wish to assign every non-edge pixel to be either *figure* or *ground*, i.e., inside the object of interest or not inside the object of interest. This will be formulated as an energy minimization problem. We will represent the labeling as a function $L(p)$, where $L(p) = 1$ when p is inside the object, and $L(p) = 0$ when p is outside of the object.

Because we are trying to prevent edges from being used multiple times, we would in particular like to prevent or penalize the use of an edge pixel in opposite directions. We thus add variables to our energy minimization problem to represent the orientation of each edge pixel. We constrain this orientation to be orthogonal to the gradient at the edge pixel, so each edge pixel has two choices of orientation, represented as a unit vector $D(p)$. We want to assign orientations so that the object of interest has its boundary traced out clockwise. We also want the orientation assignment to be consistent with the output of our global parsing algorithm.

4.5 Local Constraints

We use the output of our local interpretation procedure to modify global inference in two ways:

- A line segment that goes over an edge pixel with the wrong orientation is assigned the negative of its normal cost (corresponding to the assumption that it will be used twice in the parse with the correct orientation, and once with the reverse orientation). The first part of Figure 4.2 depicts a problematic parse that will be penalized appropriately by this heuristic.
- A line segment is disallowed if too high a fraction of its pixels are classified as interior, or if too high a fraction are classified as exterior. (Here interior and exterior refer to the output of our local inference procedure.) This is meant to eliminate short-circuiting, in which the optimization procedure double-counts an edge by taking a round-about way from its head to its tail. Unless the curve encompasses the whole object, this test generally prevents short-circuiting. The second part of Figure 4.2 shows an example of short-circuiting.

4.6 Energy functions

We formulate our energy minimization problem as the sum of four terms:

$$U(L, D) = U_{seg}(L) + U_{or}(D) + U_{int}(L, D) + U_{yd}(D),$$

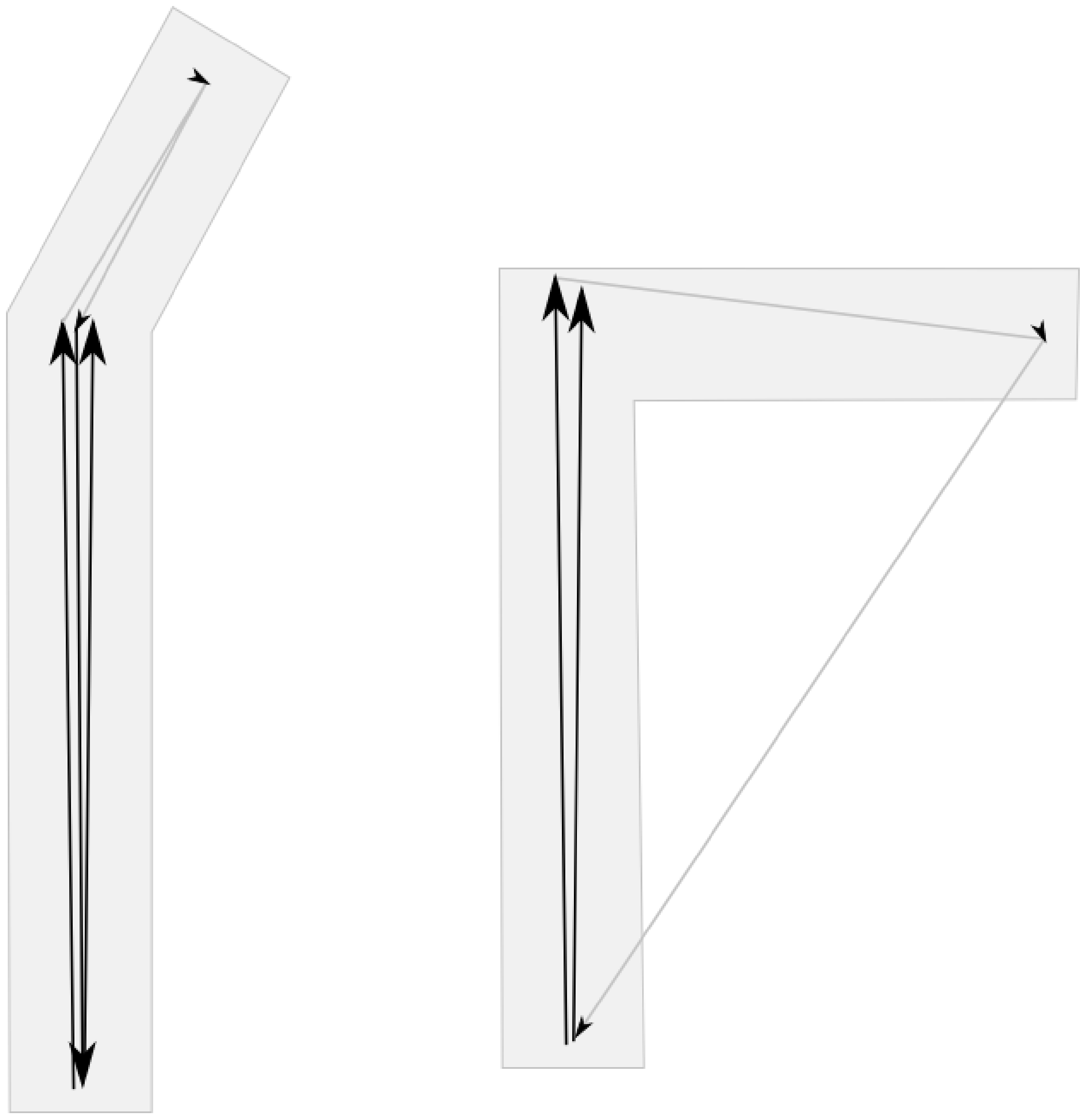


Figure 4.2: Motivation for local constraints. Grey regions represent areas where there is evidence for edges.

where U_{seg} enforces coherence in the segmentation, U_{or} enforces coherence in the orientation assignment, U_{int} enforces consistency between the segmentation and the orientation assignment, and U_{yd} enforces consistency between the orientation assignment and the previous curve selected by our global parsing algorithm. We now discuss each term in turn.

4.6.1 $U_{seg}(L)$

$$U_{seg}(L) = \sum_{p \sim q, p,q \notin E} \begin{cases} -\alpha & L(p) = L(q) \\ \alpha & \text{otherwise} \end{cases}$$

This energy function is very simple, it is just a sum over all pairs of adjacent non-edge pixels, where the cost is $-\alpha$ if the two pixels have the same label, and α if they have different labels, where $\alpha > 0$ is some constant.

This energy function is a standard model from statistical physics called the Ising model. It is commonly used in computer vision to denoise images (Besag [1993], Geman and Geman [1984]). Here we are using it to penalize non-edge boundaries between figure and ground, which should be rare. When they are necessary, this energy term will push non-edge boundaries to be short and simple.

4.6.2 $U_{or}(D)$

$$U_{or}(D) = -\sum_{p,q \in E} w_{p,q} M(p) M(q) \left[D(p) \cdot D(q) \right],$$

where $M(p)$ is the gradient magnitude at pixel p and $\cdot$ is the dot product between vectors.

This energy function pushes our orientation assignment to vary smoothly, as nearby edge pixels are penalized if their orientations do not point in the same direction. We weight this penalty by the gradient magnitude, so that stronger edges are given more influence. The function $w_{p,q}$ is a weighting that specifies how influence dies off over the distance between p and q. We have used $w_{p,q} = e^{-(x^2+y^2)/4w^2}(1 - \beta|y|)$, where x is the component of $q - p$ perpendicular to the gradient at p, and y is the component of $q - p$ parallel to the gradient at p. Note that $w_{p,q} \neq w_{q,p}$. The $(1 - \beta|y|)$ factor captures the fact that parallel edges sufficiently far apart are more likely to go in opposite directions, because they are likely to be the two sides of a part of the object.

4.6.3 $U_{int}(L, D)$

$$U_{int}(L, D) = \sum_{p \notin E} \sum_{q \in E} e^{-\frac{||q-p||^2}{\gamma^2}} (1 - 2L(p)) M(q) \sin\theta(q - p, D(q)),$$

where $M(q)$ is the gradient magnitude at q, and $\theta(u, v)$ is the angle between vectors u and v.

This energy function is motivated by the observation that the angle between $q - p$ and $D(q)$ tends to be close to 90 degrees when p is in the interior of the object and $D(q)$ is going clockwise around the object. When p is outside the object and $D(q)$ is going clockwise around the object, the angle between $q - p$ and $D(q)$ tends to be close to 270 degrees. This is shown in Figure 4.3. Multiplying the sine of this angle by $(1 - 2L(p))$, which is -1 in the interior and 1 in the exterior, yields a quantity that is negative if $L(p)$ and $D(q)$ are consistent with one another, and thus consistency is rewarded when we minimize the energy.

4.6.4 $U_{yd}(D)$

$$U_{yd}(D) = -\delta \sum_{\ell_{p,q} \in Y} \sum_{r \in \ell_{p,q}} D(r) \cdot \frac{\ell_{p,q}}{||\ell_{p,q}||},$$

where $\cdot$ is the dot product of vectors.

This energy function simply rewards orientation assignments which go in the same direction as the line segments that make up our initial parse.

4.7 Experimental Results

In our experimental results, we show the final parse chosen. Recall that we parse first, do local inference, and then reparse with modified data costs. We also show the segmentation into

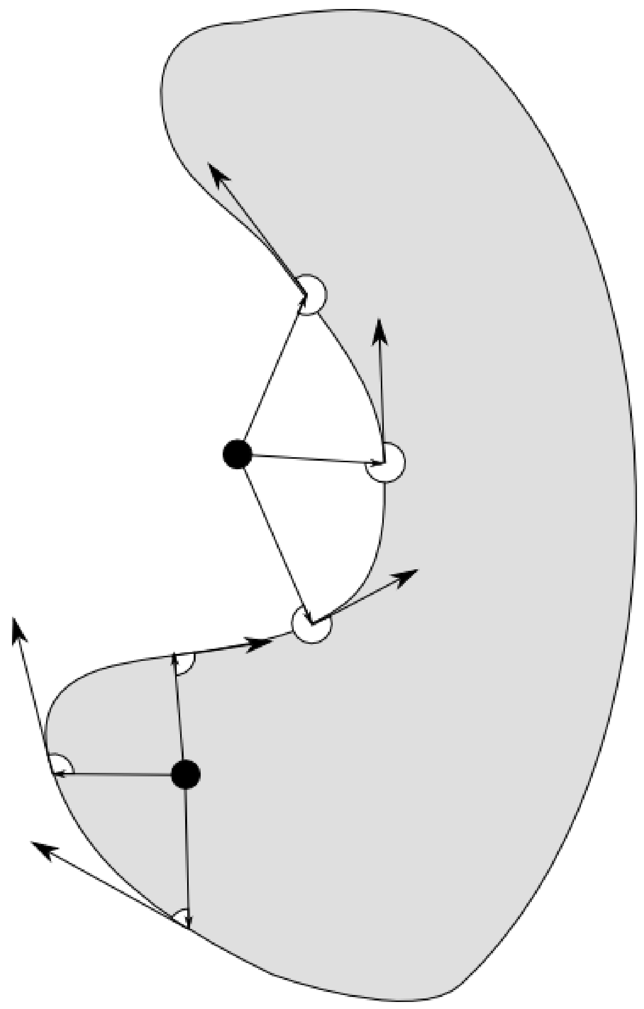


Figure 4.3: The angle between $q - p$ and $D(q)$ should be close to 90 degrees for interior p and close to 270 degrees for exterior p.

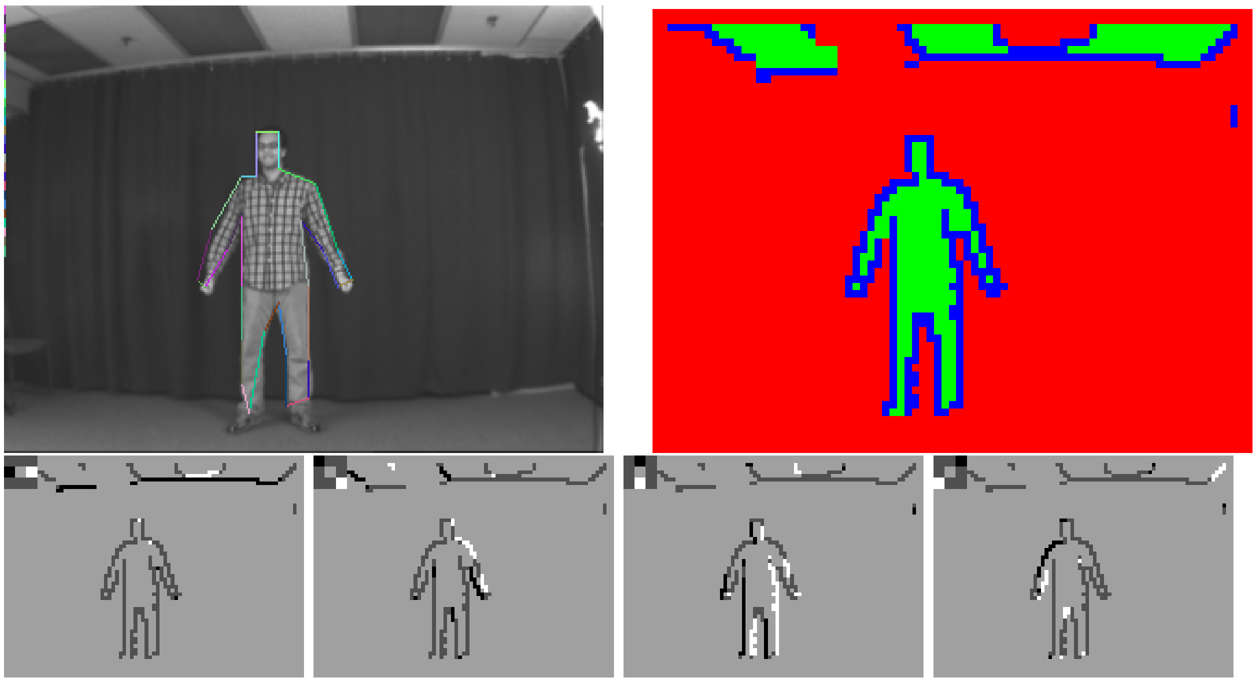


Figure 4.4: Output of detection algorithm. Final detection shown top left, segmentation shown top right, orientation assignments shown in four bottom images. The top left of the orientation images gives a key to the orientation labels.

figure and ground, and the orientation assignment selected by our local inference procedure. The orientation assignment is shown in four different images corresponding to four different pairs of opposite orientations. One orientation is shown in black, and its opposite is shown in white, while other orientations are shown in grey. The orientation depicted is shown by the squares in the top left of each image.

These results demonstrate that our local inference procedure, combined with our global parsing procedure, allows us to locate human silhouettes in clutter-free images.

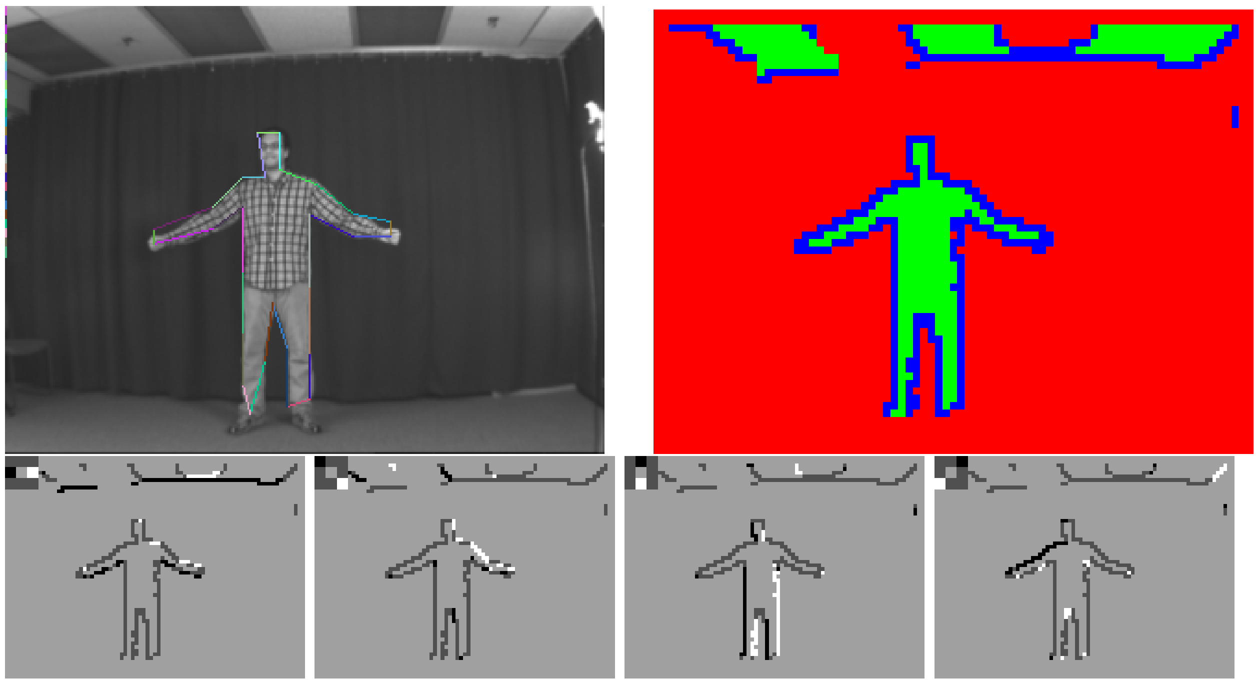


Figure 4.5: Output of detection algorithm. Final detection shown top left, segmentation shown top right, orientation assignments shown in four bottom images. The top left of the orientation images gives a key to the orientation labels.

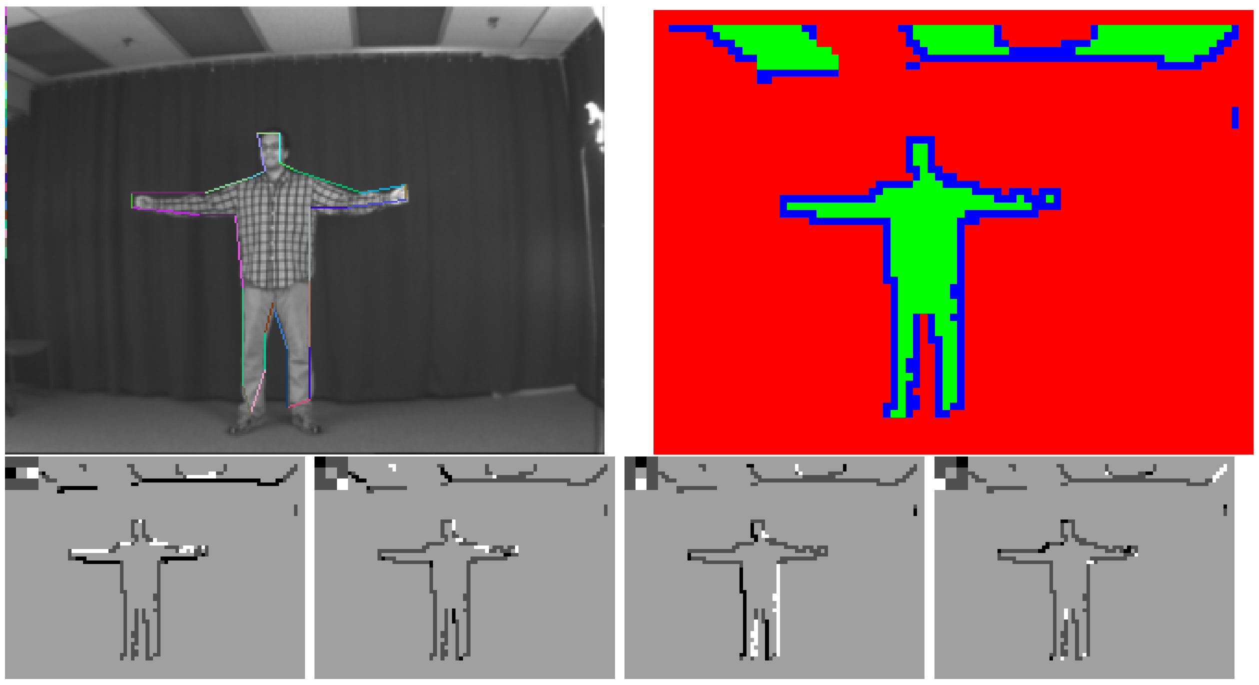


Figure 4.6: Output of detection algorithm. Final detection shown top left, segmentation shown top right, orientation assignments shown in four bottom images. The top left of the orientation images gives a key to the orientation labels.

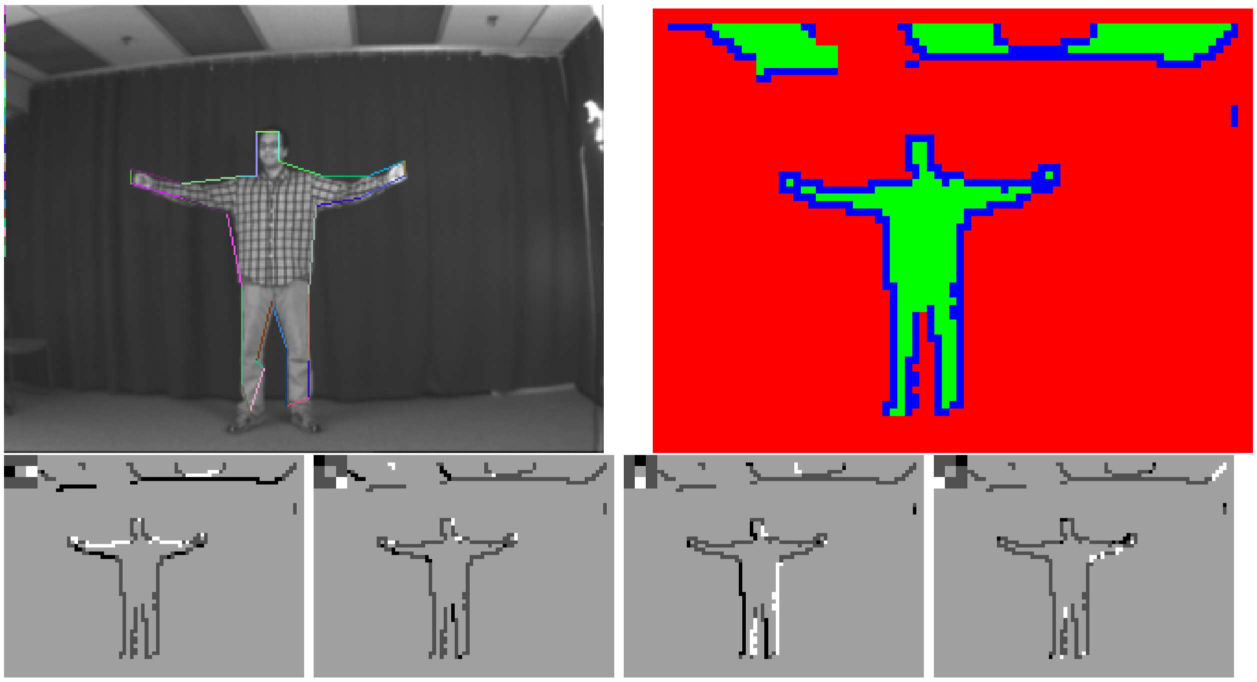


Figure 4.7: Output of detection algorithm. Final detection shown top left, segmentation shown top right, orientation assignments shown in four bottom images. The top left of the orientation images gives a key to the orientation labels.

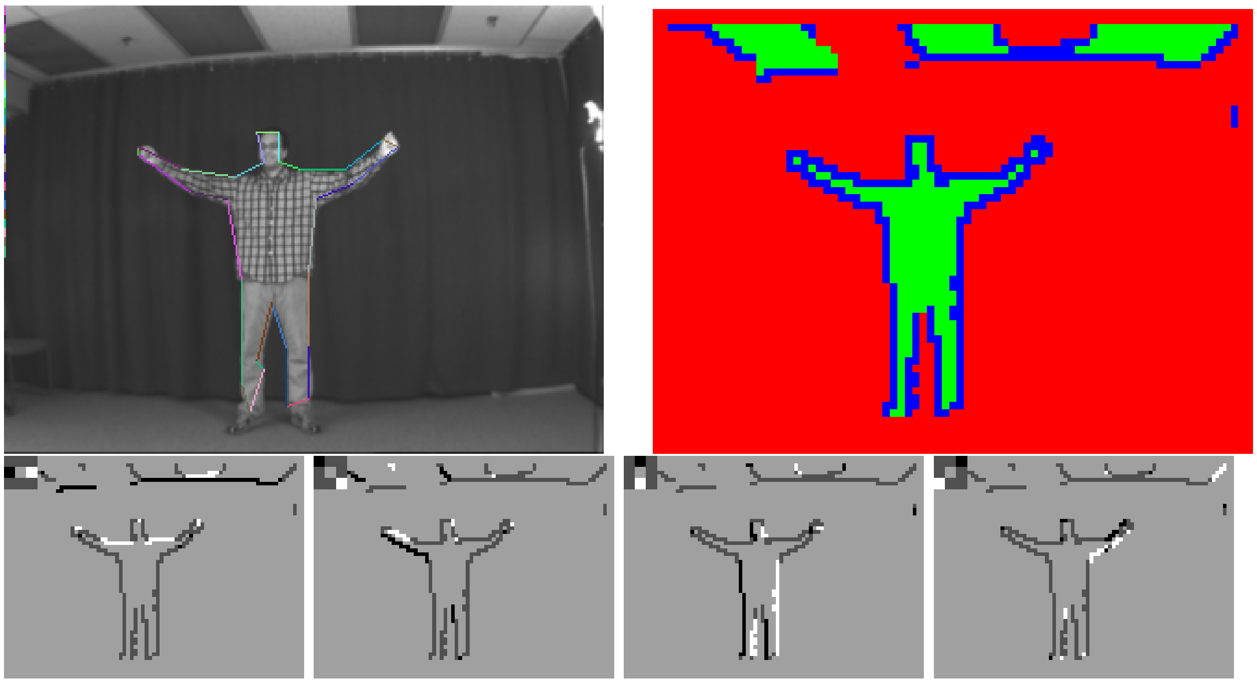


Figure 4.8: Output of detection algorithm. Final detection shown top left, segmentation shown top right, orientation assignments shown in four bottom images. The top left of the orientation images gives a key to the orientation labels.

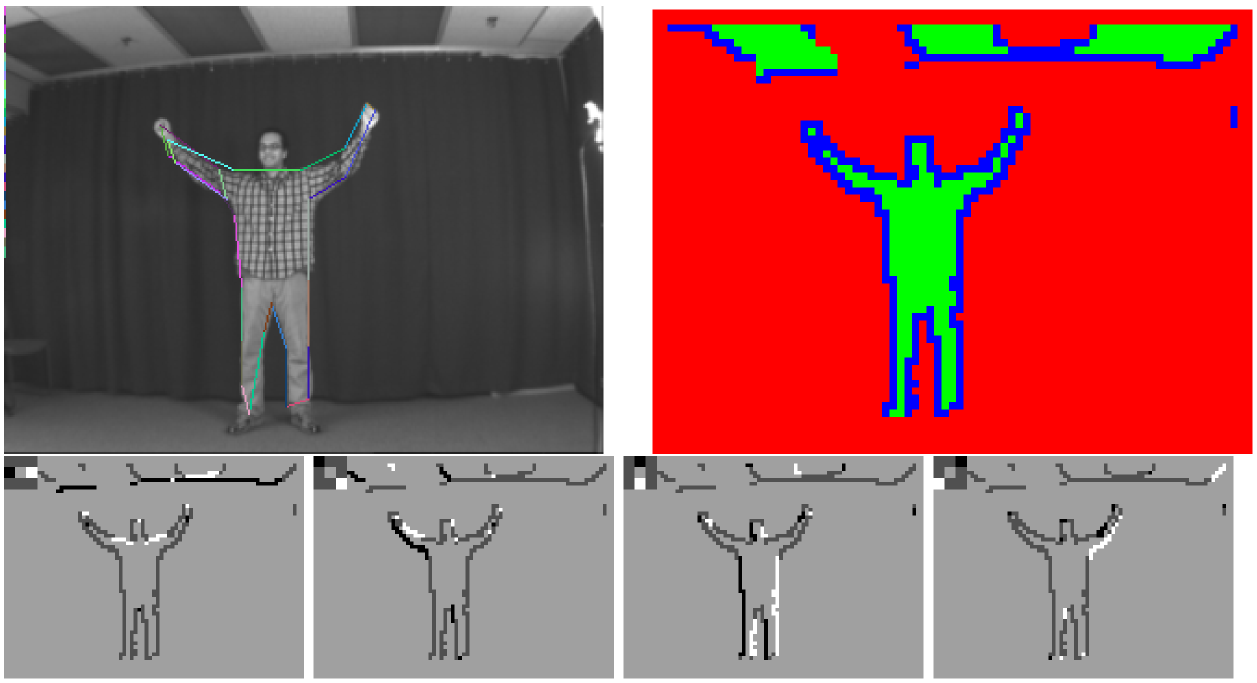


Figure 4.9: Output of detection algorithm. Final detection shown top left, segmentation shown top right, orientation assignments shown in four bottom images. The top left of the orientation images gives a key to the orientation labels.

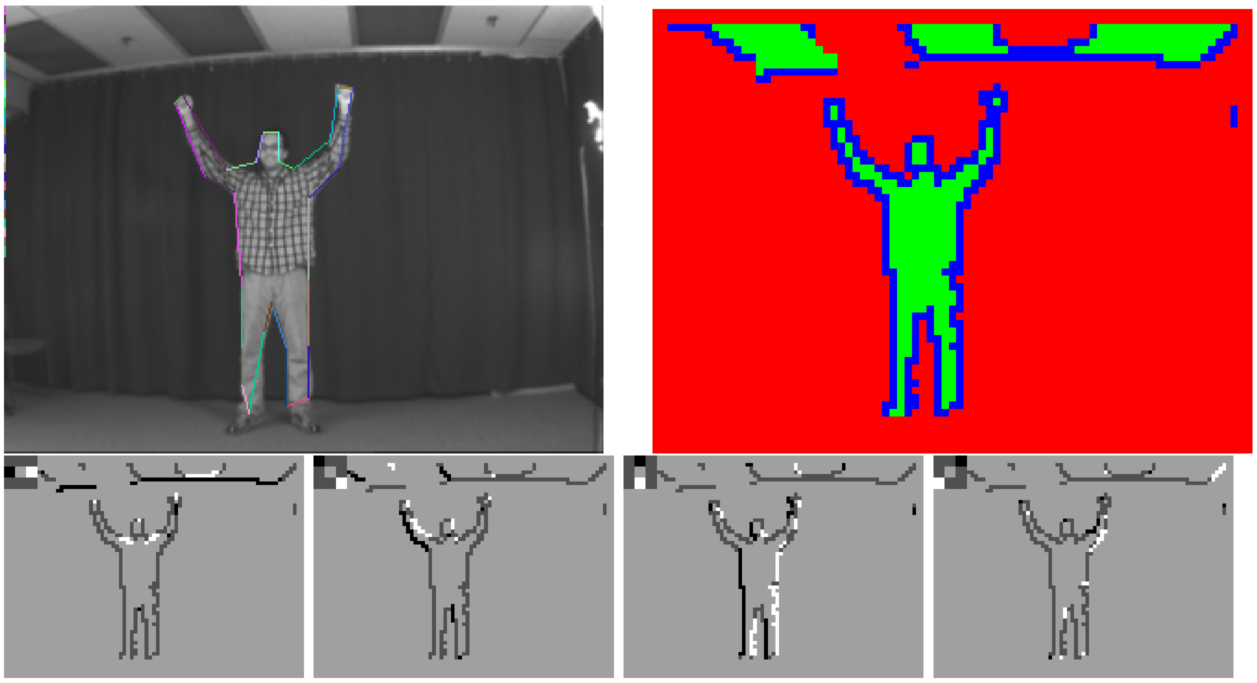


Figure 4.10: Output of detection algorithm. Final detection shown top left, segmentation shown top right, orientation assignments shown in four bottom images. The top left of the orientation images gives a key to the orientation labels.

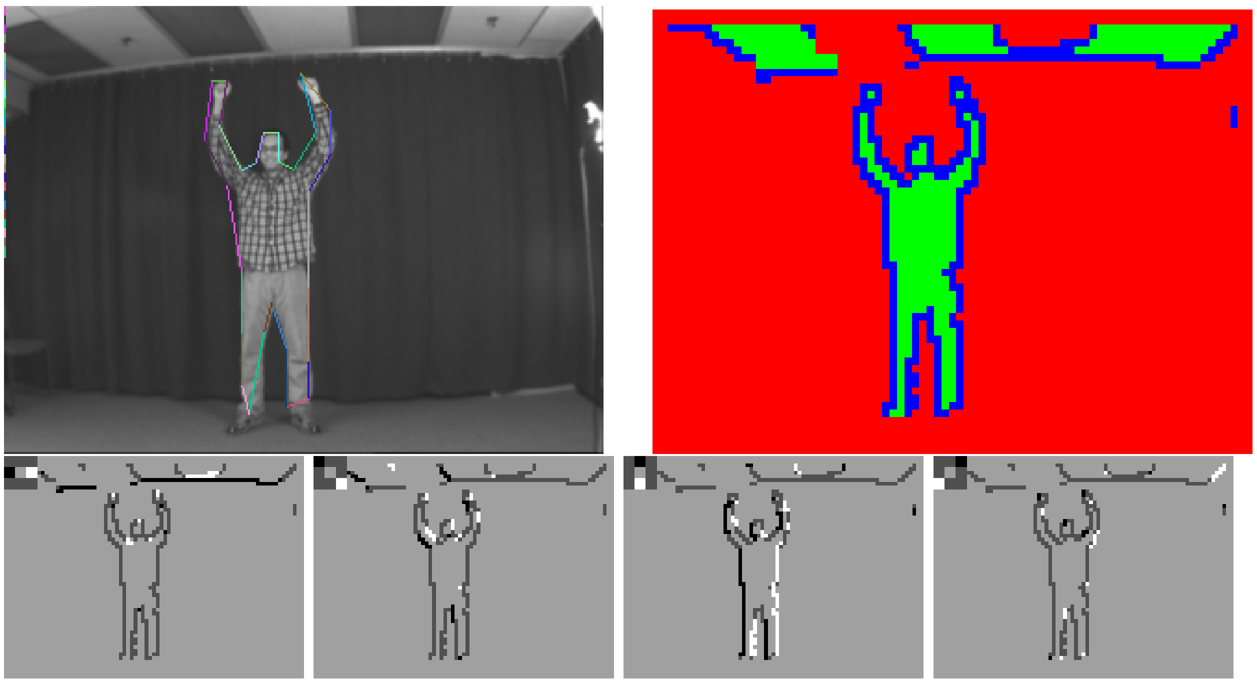


Figure 4.11: Output of detection algorithm. Final detection shown top left, segmentation shown top right, orientation assignments shown in four bottom images. The top left of the orientation images gives a key to the orientation labels.

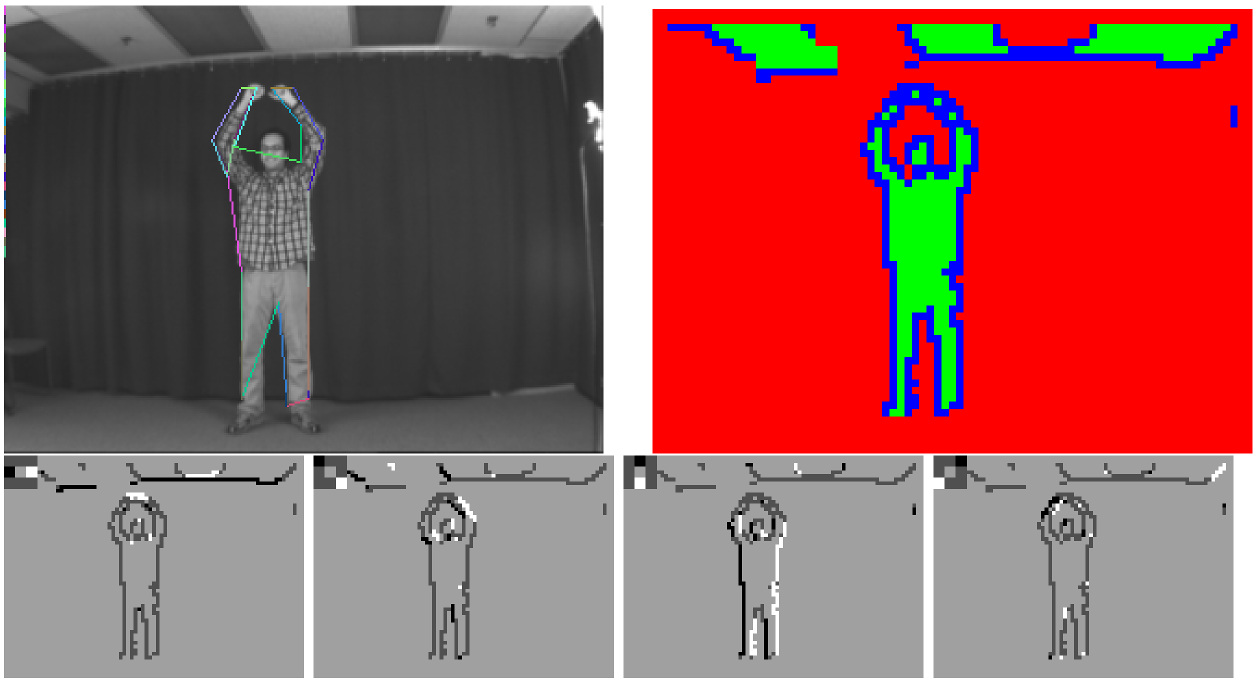


Figure 4.12: Output of detection algorithm. Final detection shown top left, segmentation shown top right, orientation assignments shown in four bottom images. The top left of the orientation images gives a key to the orientation labels.

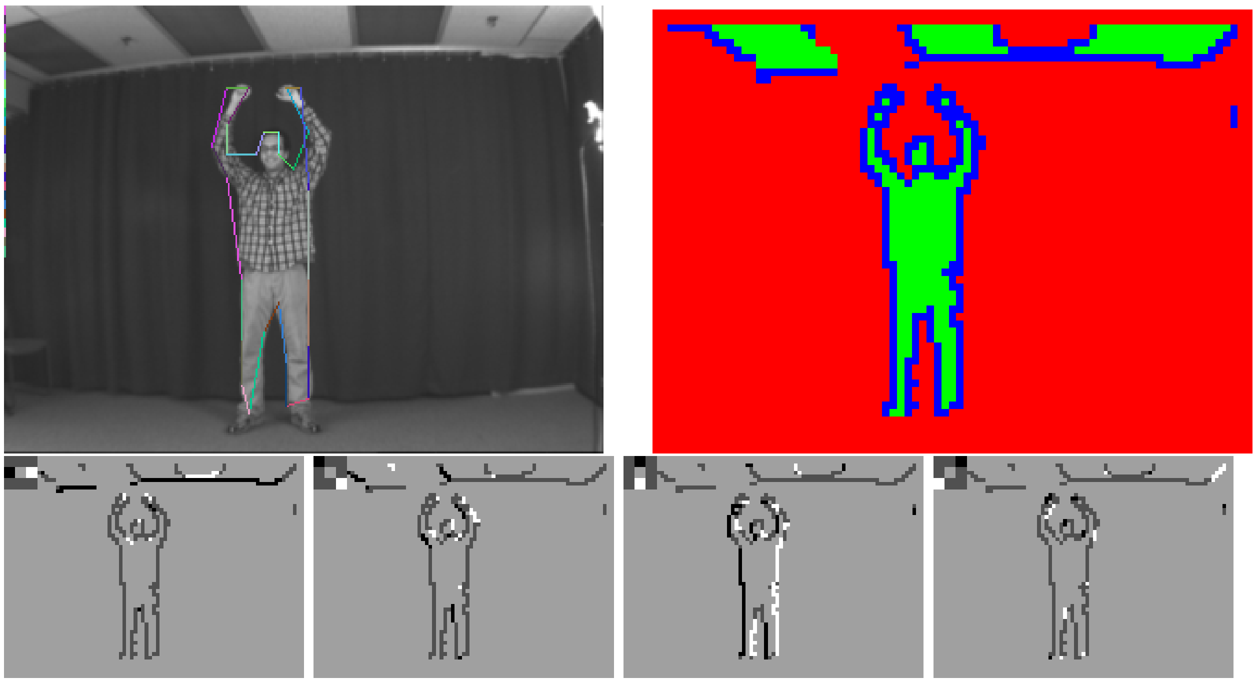


Figure 4.13: Output of detection algorithm. Final detection shown top left, segmentation shown top right, orientation assignments shown in four bottom images. The top left of the orientation images gives a key to the orientation labels.

CHAPTER 5
FUTURE WORK: STRUCTURE LEARNING FOR GRAMMATICAL MODELS

5.1 Introduction

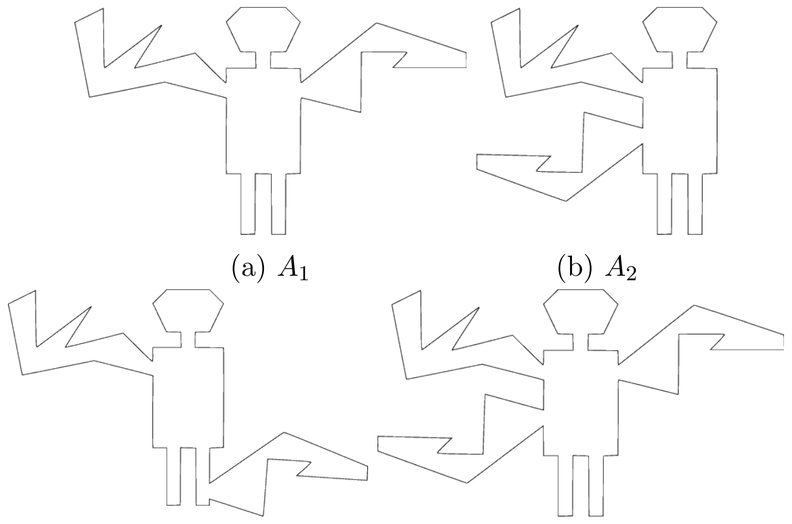
5.1.1 Structural Variation

Natural object categories such as cars and people exhibit two kinds of variation: continuous or "plastic" deformations and discontinuous structural variations. Therefore, object models which allow for both will make vision algorithms much more powerful. The potential for a satisfying account of large structural variation is one of the most intriguing possibilities of grammatical methods.

One of the simplest structural variations is occlusion: part of an object may not be visible, usually because something between the object and the camera is occluding it. Occlusion has been well understood in computer vision for a long time, and models can be made robust to it, e.g., the Hausdorff distance in Huttenlocher et al. [1993].

Another common way that objects exhibit structural variation is by having *optional parts*: a dog may or may not have a tail, a person may or may not have a hat. Occlusion models are capable of recognizing such objects with or without their optional parts, but they do not accurately model optional parts. An optional part is a particular subset of the object that is likely to not appear, while occlusion allows any not-too-large subset of the model to disappear.

The usefulness of more general structural variation can be seen in Figure 5.1. Here, the human eye notices a large similarity between the two shapes A_1 and A_2, but many curve models would see very little similarity.


(a) A_1 (b) A_2

(c) A_3 (d) A_4

Figure 5.1: If A_1 is the original curve, which other curve is most similar to it? Figure adapted from Basri et al. [1998].

We might intuitively describe the second shape, A_2, as

A_2 = "Take A_1, snap off the right appendage, and reattach it beneath the left appendage.".

This highlights several important points:

The description 5.1.1 of A_2 is very short in English, and might be even shorter in a specialized curve model encoding. Description length is a good proxy for the conditional probability of observing A_2 given that it is a distortion of A_1 (Bienenstock et al. [1997]).

Structural variation is a fundamental problem in modeling visual objects. In the absence of a practical model of structural variation, we must model variation as continuous deformation. Then, any model that declares A_1 and A_2 to be similar will think that A_3 or A_4 is even more similar to A_1.

Structural variation cannot be modeled without a semantically meaningful decomposition of the original curve, like that seen in Figure 5.2. Description 5.1.1 crucially relies on "the right appendage" making sense to the listener. Thus, perceptually simple structural variation must respect the perceived structure of the original curve. Contrast Figure 5.1 with Figure

5.3, where a similar transformation has been applied with no regard to the perceived structure of the original curve.

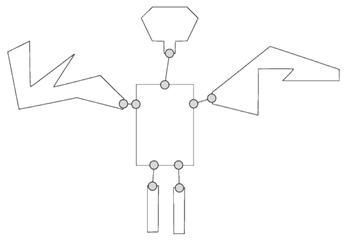


Figure 5.2: The original shape from Figure 5.1, decomposed into semantically meaningful parts. We argue that this decomposition explains why the variation in Figure 5.1 is less semantically different than the variation in Figure 5.3. Adapted from Basri et al. [1998].

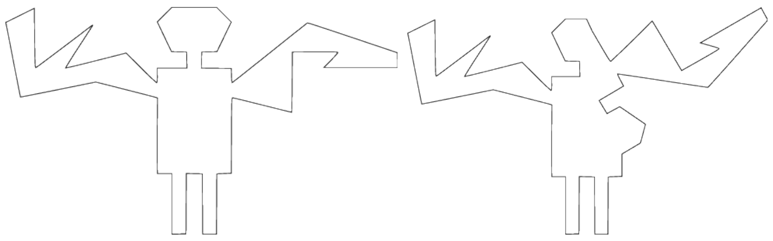


Figure 5.3: Two shapes which are not perceptually very similar, although they are related by a transformation as simple as that in Figure 5.1. The problem is that the transformation does not respect the perceived structure of the original. Adapted from Basri et al. [1998].

Mixture models are a class of models that do not suffer from the continuous deformation problem of Figure 5.1. However, if there are multiple independent structural variations possible, it is unlikely that we will see every combination of each form. Consider the shape grammar $\mathcal{G}_n$ that generates shapes that have n arms, each of which can take either of two forms:

$$S \to \underbrace{Z \dots Z}_{n}$$

$$Z \to A$$

$$Z \to B,$$

where A is pointy and B is rectangular. We show four shapes possible under this grammar in Figure 5.4. A classic mixture model will not be able to generalize in this scenario without

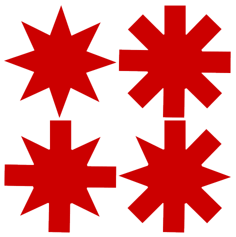


Figure 5.4: Four shapes from $\mathcal{G}_8$.

exponentially many training examples, since there are 2^n possible shapes. If we instead have mixture models at the level of individual structural variations, then our model is a grammatical model in the style of Section 1.1.

5.2 Searching Over Grammars

We want to learn a grammar that explains our training data well, but we also want a simple grammar, because simpler models exhibit better generalization. We can reward simple grammars by using a prior based on the Minimum Description Length, described in the next section. We can then attempt to maximize the posterior probability of $\mathcal{G}$ given the training data and this prior.

Finding the optimal such grammar is a very hard unsolved problem even for standard CFG's, as discussed in Section 1.6.1. In particular, we do not know of any optimal way to search over grammatical structures, so we propose to start from the initial grammars described in Section 2.7 and proceed by greedily applying transformations such as replacement (discussed in Section 5.4), merging (discussed in Section 5.5), and rule deletions. We could also try a slightly more flexible search algorithm, such as the beam search used in Stolcke [1994].

In the light of Stolcke [1994], one should consider an initial structure and transformations such that we can reach any grammatical structure from the initial one.

To assess such a search strategy, a simple task would be to build a shape grammar by hand and take samples from it, and try to recover the grammar from the samples. It is unlikely that we will recover the same exact grammar, so we would instead try to show that we can improve some measure of closeness, for instance the cross-entropy on unseen samples.

5.3 MDL Priors over Grammar Structure

5.3.1 MDL Priors over Grammar Structure

Let $\mathcal{G} = (\mathcal{N}, \mathcal{R}, \mathcal{S}, \ell, \mathcal{M}, \mathcal{X})$ be a grammar. We wish to specify a prior over the structural parts of $\mathcal{G}$: $\mathcal{N}, \mathcal{R}, \mathcal{S}$. We use a minimum description length prior for the structure of the grammar:

$$P(\mathcal{N}, \mathcal{R}, \mathcal{S}) = \frac{1}{Z} e^{-len(enc(\mathcal{N}, \mathcal{R}, \mathcal{S}))}$$

for some binary encoding function $enc(\cdot) : \{\mathcal{G}\} \to \{0,1\}^*$. enc must be uniquely decodable, i. e., $enc(\mathcal{N}, \mathcal{R}, \mathcal{S}) = enc(\mathcal{N}', \mathcal{R}', \mathcal{S}') \implies \mathcal{N} = \mathcal{N}', \mathcal{R} = \mathcal{R}', \mathcal{S} = \mathcal{S}'$.

The encoding function enc should be as concise as possible, so that simple grammars are rewarded for their simplicity. We choose to represent $\mathcal{N}, \mathcal{R}$, and $\mathcal{S}$ as

$$(|\mathcal{N}|, \{(i_X, i_Y, i_Z) \mid [X \to YZ] \in \mathcal{R}\}, \{(i_Y, i_Z) \mid YZ \in \mathcal{S}\}),$$

where the i_X denote some consistent numbering of the nonterminals. We assume that every nonterminal $X \in \mathcal{N}$ has the rule $[X \to \ell] \in \mathcal{R}(X)$, and thus do not explicitly encode this information.

Each component of this representation can be encoded in binary:

- $|\mathcal{N}|$ is represented as a standard binary integer, taking up $k_{\mathcal{N}} \lceil \log_2 \mathcal{N} \rceil$ bits
- Each (i_X, i_Y, i_Z) is represented as three binary integers, each zero-padded to take up exactly $k_{\mathcal{N}}$ bits.
- Each (i_Y, i_Z) is represented as two binary integers, each zero-padded to take up exactly

$k_{\mathcal{N}}$ bits.

We thus use up $k_{\mathcal{N}}(1 + 3|\mathcal{R}| + 2|\mathcal{S}|)$ bits. We also need to use $2\lceil \log_2 k_{\mathcal{N}} \rceil + 2$ bits to specify how to break the stream of bits into "words" of $k_{\mathcal{N}}$ bits. (We simply need to encode $k_{\mathcal{N}}$ itself. We need to be able to specify where the encoding of $k_{\mathcal{N}}$ stops and the rest of the encoding begins. One relatively simple way to do this is to encode $k_{\mathcal{N}}$ in binary, and then preface every digit with a 1. We then use 00 to denote the end of $k_{\mathcal{N}}$.)

Thus our total description length is

$$\lceil \log_2 |\mathcal{N}| \rceil (1 + 3|\mathcal{R}| + 2|\mathcal{S}|) + 2 \lceil \log_2 \log_2 |\mathcal{N}| \rceil + 2$$

5.4 Creating Reusable Parts with Replacement

Given a grammar $\mathcal{G}$, we wish to create a grammar which still explains our training data, and is simpler. We would also like to create grammars that have reusable parts, as explained in Bernstein and Amit [2005]. This can be done using the operation of *replacing Y with X*: whenever Y is on the right-hand side of a rule, we replace it with X. Whenever Y is on the left-hand side of a rule, we delete that rule. This may lead to a merging of two rules $[Z_1 \to XZ_2], [Z_1 \to YZ_2]$, in which case we combine the multinomial probabilities and average the midpoint distributions.

Why does this operation create reusable parts? Ideally, before the replacement, Y and X are nonterminals that are used to parse similar curves (i.e., they have similar yields), but which fit differently into the larger grammar. After the replacement, these curves will be parsed only with X, and X will fit into the larger grammar in two different ways. X has thus become a reusable part.

Note that replacement is asymmetric. This could be good; since approximate similarity is not transitive, we would like to replace nonterminals $X_1, \ldots, X_k$ with whichever X_i is most representative.

This transformation should be applied when the nonterminals X and Y have similar *internal* structure. Intuitively, this means that any curve generated by X can be generated by Y with a similar probability, and vice versa.

Suppose that we have two nonterminals X and Y, and we want to decide whether we should replace Y with X, given that we want a simple model which still explains our data, which is formalized by trying to maximize the posterior probability of the chosen grammar. Performing the replacement will simplify the grammar, which will be reflected in the MDL prior over structure. In order to make sure that we are still explaining the data, we need to consider the ratio of the likelihood under the current grammar to the likelihood under the grammar after performing the replacement.

Consider the quantity

$$Q(\mathcal{G}') = E_{\mathbf{T}\sim\mathbf{T}|\mathbf{C},\mathcal{G}}\left[\log P(\mathbf{C},\mathbf{T} \mid \mathcal{G}')\right] + \log P(\mathcal{G}')$$

used as a measure of grammar quality in the EM algorithm. Consider the difference in $Q(\mathcal{G}')$ created by appling a replacement transformation:

$$\begin{aligned} Q(\mathcal{G}') - Q(\mathcal{G}) =& E_{\mathbf{T}\sim\mathbf{T}|\mathbf{C},\mathcal{G}'}\left[\log P(\mathbf{C},\mathbf{T} \mid \mathcal{G}')\right] \qquad - E_{\mathbf{T}\sim\mathbf{T}|\mathbf{C},\mathcal{G}}\left[\log P(\mathbf{C},\mathbf{T} \mid \mathcal{G})\right] \\ & + \log P(\mathcal{G}') - \log P(\mathcal{G}). \end{aligned}$$

The last two terms correspond to the reward for simplifying the grammar. The first term measures how much better or worse the new grammar is at explaining the data.

We can make the simplifying assumption or approximation (used in Stolcke [1994]) that the distribution $\mathbf{T} \sim \mathbf{T}|\mathbf{C},\mathcal{G}'$ is arrived at by sampling from $\mathbf{T} \sim \mathbf{T}|\mathbf{C},\mathcal{G}$ and performing the operation Replace(Y,X) on each parse tree; we then have

$$
\begin{aligned}
Q(\mathcal{G}') - Q(\mathcal{G}) &= E_{\mathbf{T}\sim\mathbf{T}|\mathbf{C},\mathcal{G}}\left[\log P(\mathbf{C},\mathbf{T} \mid \mathcal{G}') - \log P(\mathbf{C},\mathbf{T} \mid \mathcal{G}\right] + \log P(\mathcal{G}') - \log P(\mathcal{G}) \\
&= E_{\mathbf{T}\sim\mathbf{T}|\mathbf{C},\mathcal{G}}\left[\sum_{Y_{ij}\in T} \log \frac{P(X_{ij})}{P(Y_{ij})}\right] + \log P(\mathcal{G}') - \log P(\mathcal{G}) \\
&= \sum_{a,i,j} Count_a(Y_{ij}) \log \frac{P(X_{ij})}{P(Y_{ij})} + \log P(\mathcal{G}') - \log P(\mathcal{G})
\end{aligned}
$$

5.4.1 Approximating the KL Divergence Efficiently

We can also arrive at a similar but slightly different formula by trying to empirically estimate the KL divergence between the *inside distributions* of X and Y. The inside distribution of X is the distribution over curves that comes from sampling in the standard way. We call this distribution the inside distribution because its probabilities are calculated in the first pass of the Inside-Outside Algorithm, as explained in Section 2.6.

The Kullback-Leibler divergence is a popular choice for measuring the distance between two distributions. It is defined as

$$KL(P||Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}.$$

We can rewrite this as

$$KL(P||Q) = \mathbb{E}_{x\sim P(\cdot)}\left[\log \frac{P(x)}{Q(x)}\right],$$

and it is clear that we can approximate it as

$$KL(P||Q) \approx \frac{1}{n}\sum_i \left[\log \frac{P(x_i)}{Q(x_i)}\right],$$

where $x_1, \ldots, x_n$ are samples from P. If $x_1, \ldots, x_n$ are instead samples from a distribution

R, then we can approximate

$$
\begin{aligned}
KL(P||Q) &= \sum_x R(x)\frac{P(x)}{R(x)}\log\frac{P(x)}{Q(x)}. \\
&= \mathbb{E}_{x\sim R(\cdot)}\left[\frac{P(x)}{R(x)}\log\frac{P(x)}{Q(x)}\right] \\
&\approx \frac{1}{n}\sum_i\left[\frac{P(x_i)}{R(x_i)}\log\frac{P(x_i)}{Q(x_i)}\right].
\end{aligned}
$$

When $R()$ is the empirical distribution $R(x_i) = \frac{1}{n}\#\{j \mid \ x_j = x_i\}$ this simplifies to:

$$
\approx \sum_i\left[P(x_i)\log\frac{P(x_i)}{Q(x_i)}\right].
$$

Let X be a nonterminal. For every subcurve C of any training curve, we can use reweighting to consider C as a sample from X's inside distribution. In this case, P, Q are the inside distributions of X and Y, respectively. This approximation is especially appealing because the Inside-Outside algorithm gives us a very efficient way to simultaneously compute $P_X(C_a[i:j])$ for every nonterminal X and every subcurve $C_a[i:j]$ of every training curve C_a. Specifically, we can compute all these in $O(kn^3|\mathcal{N}|)$ time, where k is the number of training curves, and n is the maximum length of a training curve.

When dealing with the initial grammar, this approximation should be reasonable, since our initial grammars don't generate any curves that are too dissimilar from our training curves. We hope that generalization will take place as the grammar is simplified; thus, the approximation will get worse as we simplify the grammar.

Given training data $C_1, \dots, C_n$, we can evaluate any transformation that takes $\mathcal{G}$ to $\mathcal{G}'$ by examining the posterior

$$
P(\mathcal{G}'|C_1, \dots, C_n) = P_{prior}(\mathcal{G}')P(C_1, \dots, C_n|\mathcal{G}'),
$$

which is computable via parsing. This gives a general search strategy: simply pick the transformation that increases the value of the posterior the most.

5.5 Merging to Create Factorization

5.5.1 Goal: Merging and Factorization

We would like to show that grammars can capture an exponential amount of variability through factoring, as discussed in Section 5.1.

The easiest task would be to correctly apply merging (Section 5.5) to the n-armed shapes of Section 5.1, so that we can generate all of the shapes in the class after having seen a small number of them. We can make this easier by having different arm variants in each of the n arm slots. We can also start on an easier version by training on bracketed samples.

A harder merging task is to use merging to get accurate models of the Romer and Weizmann Horse datasets.

5.5.2 Creating Choice with Merging

We want to create grammars that have choice, as described in Section 5.1.1. This can be done by the operation of *merging X and Y*: we replace every instance of Y in either the left or right-hand side of a rule with X. This may lead to a merging of two rules $[X \to Z_1 Z_2], [Y \to Z_1, Z_2]$ (and some other cases). When this happens, we combine the multinomial probabilities and average the midpoint distributions of the rules.

As a heuristic to guide merging, we plan to apply the approach of the last section, but to use the outside probability instead of the inside probability. If two nonterminals X and Y have similar outside probabilities on every subcurve, then X and Y can be said to have similar *outer structure*, since the grammar is placing them in similar contexts. Intuitively, this means that X and Y are interchangeable, so that for any curve in which X is used to explain a subcurve, there is another equally likely curve where that subcurve has been replaced with a subcurve that can be explained by Y.

5.6 How do we Identify Natural Constituents?

5.6.1 Goal: Learning Constituents

The EM algorithm with a sparsifying prior will hopefully be able to recover grammatical structure. Therefore, if we start from a grammar with choice (see Section 3.7), we hope that doing EM iterations will give us a grammar with fewer choices that correspond to natural decompositions of shapes. Note that this method can only work with many training examples, and cannot be used to identify natural decompositions of a single shape.

Easy tasks for this would be trying to find constituents in stars and polygons, and the n-armed shape. We could also try to get intuitively reasonable constituents for silhouettes of people and horses.

5.6.2 Constituency Cues

In Basri et al. [1998], Richards et al. [1985], Kimia et al. [2003], it is argued that some shapes have natural constituents, and some shapes do not. Some cues that we believe would give us some clues to structure:

- Large-scale curvature. If we smooth the points of a curve with a Gaussian, then we can measure curvature at different levels. We feel that large-scale curvature is likely to occur only at constituent boundaries.

- Protuberances. If the curve goes out from a point x and then returns to a point y, where the length of the curve is significantly larger than the distance between x and y, then the curve from x to y is likely to be a natural constituent. This will also be true of any points close to x and y, so we should restrict ourselves to pairs (x, y) for which the ratio
$$\frac{\max_z ||x - z|| + ||y - z||}{||x - y||}$$
is at a local maximum.

5.6.3 Figure out optimal single-example grammar

We use explicit correspondences to learn the statistically best set of constituents when building a grammar from a single example. (This experiment is closely related to an experiment in Felzenszwalb [2003].) Specifically, since the points on each curve from the hand-annotated Romer dataset correspond semantically, we can ask which decomposition of the curves yields the model which gives the highest log-likelihood to the curves seen. This can be done with a simple dynamic program:

$$\begin{aligned} COST[i,k] &= \min_j COST[i,j] + COST[j,k] \\ &\quad - \min_{\mu,\kappa} \sum_a \log Watson(C_a[i], C_a[j], C_a[k]; \mu, \kappa) \end{aligned}$$

for $k \neq i, (i+1) \mod n$, and

$$COST[i, (i+1) \mod n] = 0$$

Minimizing the sum of negative log-likelihoods of the Watson distribution is simply maximum likelihood estimation for the Watson distribution, which is explained in Subsection 3.1.5.

The constituents selected by the algorithm are shown in Figure 5.5 The constituents that seemed most intuitive to me are shown in Figure 5.6.

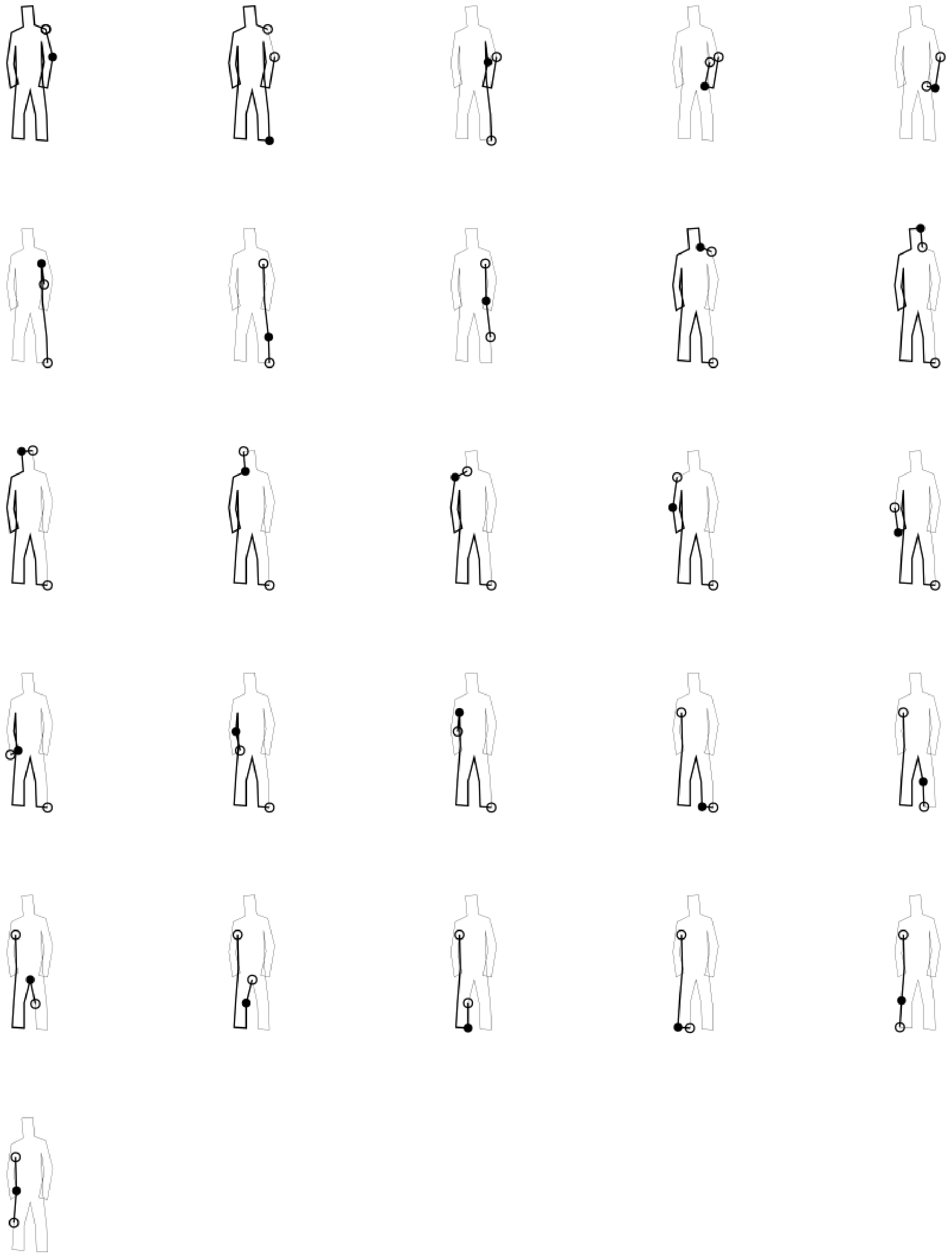


Figure 5.5: Finding optimal constituents. This figure generated by the experiment experiments/6.structure/constituents.

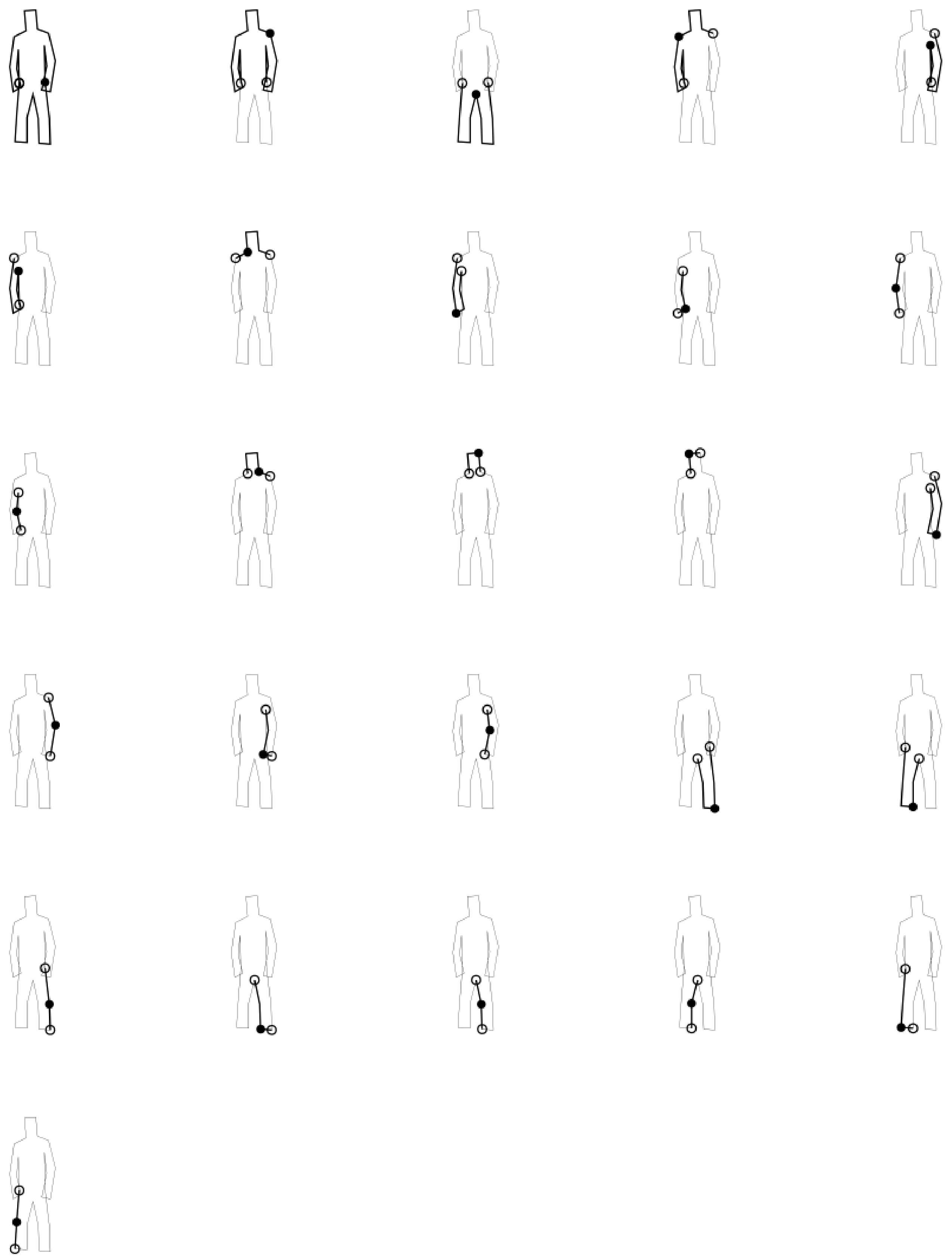


Figure 5.6: Hand-picked constituents.

CHAPTER 6
APPROXIMATE PARSING OF ONE-DIMENSIONAL SIGNALS

Recall that the problem of parsing curves with a shape grammar is closely related to the problem of parsing strings with a weighted context free grammar. The running time of context-free parsing with a standard algorithm like CKY is cubic, and it is not known how to do better than this in practice. This makes parsing prohibitively slow for long strings in particular. This prevents the use of context-free grammars in modeling long one-dimensional signals, which are instead commonly modeled with Markov models.

The main contribution of this chapter is to introduce an approach for efficient context-free parsing of very long strings. This would allow fast curve parsing with our shape grammars, and also the use of context-free models for other problems that involve long one-dimensional signals.

6.1 CKY parsing

We now recall the CKY algorithm for parsing strings with a weighted context-free grammar in Chomsky Normal Form, which is shown in Algorithm 7. Let s be a string of length n. Let $(\mathcal{X}, \mathcal{R})$ be a weighted context-free grammar, with $\mathcal{X}$ a set of symbols and $\mathcal{R}$ a set of weighted rules. Let $w(X \to \lambda)$ the weight of the rule $(X \to \lambda) \in \mathcal{R}$.

Let $[i, j)$ be the interval starting with i and ending with $j - 1$. The algorithm works by computing a cost $COST[X, [i, j)]$ for each symbol X and each interval $[i, j)$, corresponding to the cost of parsing the substring $s[i] \dots s[j-1]$ with the nonterminal X.

The $O(n^3)$ runtime of the CKY algorithm arises because there are $O(n^2)$ substrings, and many substrings can be decomposed in $O(n)$ ways. If we could limit the number of constituents and decompositions, we could speed up the algorithm. This brings us to the next section, where we define a framework for considering restricted families of constituents

Algorithm 7 CKY parsing. Returns cost of best parse.

```
Input: string $s$, context-free grammar $(\mathcal{X}, \mathcal{R})$
  for $X \in \mathcal{X}$ do
    for $i = 0$ to $n - 1$ do
      $COST[X, [i, i+1)] \leftarrow w(X \to s[i])$
    end for
  end for
  for $\ell = 2$ to $n - 1$ do
    for $i = 0$ to $n - 1$ do
      $COST[X, [i, i+\ell)] \leftarrow \infty$
      for $j = i + 1$ to $i + \ell - 2$ do
        for $X \to YZ \in \mathcal{R}$ do
          $COST[X, [i, i+\ell)] \leftarrow \min\{COST[X, [i, i+\ell)], w(X \to YZ) + COST[Y, [i, j)] +$
          $COST[Z, [j, i+\ell)]\}$
        end for
      end for
    end for
  end for
  return $COST[S, [0, n)]$
```

and decompositions.

6.2 Decomposition Families

Definition 6.2.1. Let s be a string of length n. A *decomposition family for s* is a pair $(\mathcal{I}, \mathcal{D})$, where $\mathcal{I}$ is a set of intervals $[i, j)$, and $\mathcal{D}$ is a set of decompositions of the form

$$[i, j) \to [i, k) + [k, j),$$

where $[i, j), [i, k), [k, j) \in \mathcal{I}$. $\mathcal{I}$ must contain the interval $[0, n)$.

The most trivial decomposition family is the set of all intervals $[i, j)$ such that $0 \leq i < j \leq n$, together with all possible decompositions of each $[i, j)$ into $[i, k) + [k, j)$. We will call this the *complete decomposition family.*

We would like small decomposition families that can approximate arbitrary parses reasonably well. We capture this notion with θ-flexibility:

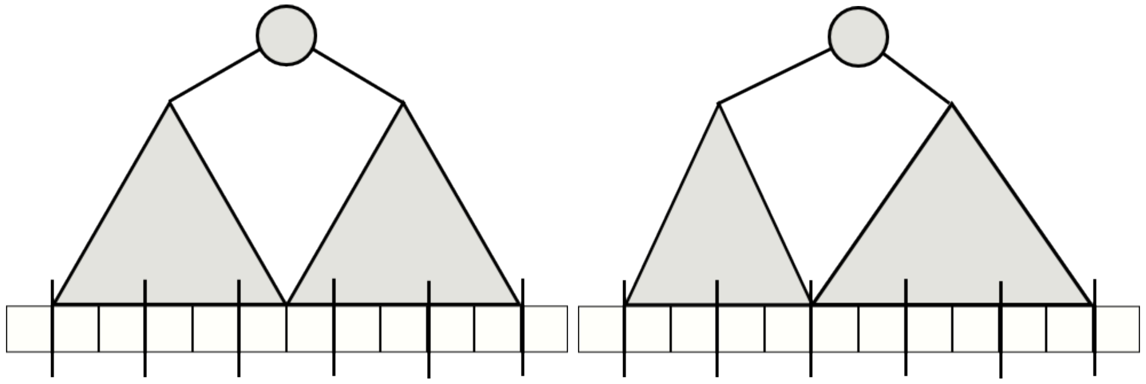


Figure 6.1: With a θ-flexible decomposition family, any parse can be adjusted to an allowable parse by moving the midpoint of each binary rule slightly. On the left, a parse before adjustment. On the right, after the adjustment. Vertical lines denote allowable midpoint choices.

Definition 6.2.2. A decomposition family $\mathcal{F} = (\mathcal{I}, \mathcal{D})$ is θ-flexible if, for every interval $[i, j) \in \mathcal{I}$, and every k such that $i < k < j$, there is a decomposition

$$[i, j) \to [i, k') + [k', j) \in \mathcal{D},$$

where

$$|k - k'| < \theta |j - i|.$$

A θ-flexible decomposition family is desirable because, for any binary parse tree, there is a similar parse tree that is allowable under the decomposition family. Here, similar means that, as we recursively subdivide a curve into subcurves, we can always choose a midpoint whose index differs from a given midpoint by at most a fixed percentage (e.g., 10% when $\theta = \frac{1}{10}$ or 5% when $\theta = \frac{1}{20}$) of the length of the subcurve, as shown in Figure 6.1. A θ-flexible decomposition family thus approximates the complete decomposition family, in some sense.

Another notion of approximation is θ-completeness:

Definition 6.2.3. Let s be a string of length n. Let $\mathcal{F} = (\mathcal{I}, \mathcal{D})$ be a decomposition family for s. An interval $[i, j)$ is called *reachable* if there is a sequence of intervals x_t such that $x_0 = [0, n), x_k = [i, j)$, and there is a decomposition with x_t on the left-hand side and x_{t+1}

on the right-hand side for every t between 0 and $k-1$.

Definition 6.2.4. A decomposition family $\mathcal{F} = (\mathcal{I}, \mathcal{D})$ is θ-complete if, for every interval $[i, j), 0 \leq i < j \leq n$, there is a reachable interval $[i', j') \in \mathcal{I}$ such that

$$|i - i'| < \theta|j - i|$$

and

$$|j - j'| < \theta|j - i|.$$

Theorem 6.2.5. *Let $\mathcal{F} = (\mathcal{I}, \mathcal{D})$ be a θ-flexible decomposition family for $\theta \leq \frac{1}{4}$. Then $\mathcal{F}$ is 2θ-complete.*

Proof. Let $[i, j)$ be an interval. Starting with the interval $[0, n)$, we build a sequence of intervals by picking midpoints so that $[i, j)$ is wholly contained in one side or the other each time. At each step, we then take the interval containing $[i, j)$ as the next interval in our sequence.

When we can no longer do this, we have an interval $[k, \ell)$ such that $k \leq i < j \leq \ell$. We claim that

$$k - i < 2\theta(j - i)$$

and

$$\ell - j < 2\theta(j - i),$$

which is what we need for 2θ-completeness.

Since we can no longer pick a midpoint without it being between i and j, by θ-flexibility, we must have

$$i - k < \theta(\ell - k)$$

and

$$\ell - j < \theta(\ell - k).$$

Furthermore, we have

$$\begin{aligned}
\ell - k &= (\ell - j) + (j - i) + (i - k) \\
&< (j - i) + 2\theta(\ell - k) \\
(1 - 2\theta)(\ell - k) &< j - i,
\end{aligned}$$

and, since $\theta \leq \frac{1}{4}$,

$$\frac{1}{2}(\ell - k) < j - i.$$

We thus have

$$i - k < 2\theta(j - i)$$

and

$$\ell - j < 2\theta(j - i),$$

as desired. □

6.3 Parsing with decomposition families

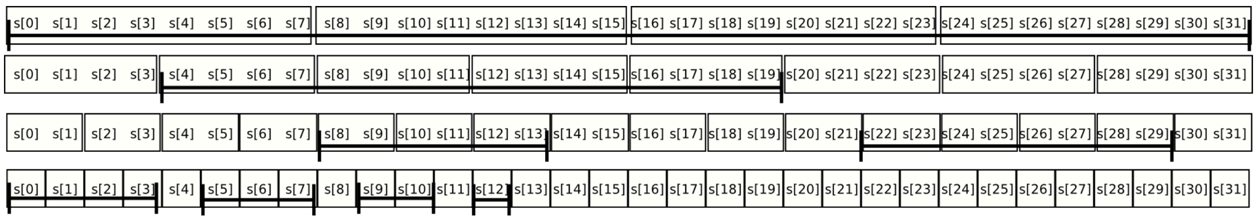
Decomposition families are interesting because they allow us to speed up parsing by searching over a restricted set of parses.

Theorem 6.3.1. *Let G be a context-free grammar with k rules. Let s be a string of length n. Let $\mathcal{F} = (\mathcal{I}, \mathcal{D})$ be a decomposition family for s. Then there is a dynamic programming algorithm (Algorithm 8) to approximately (relative to $\mathcal{F}$) parse s in time $O(|\mathcal{D}|k)$.*

Traditional parsing corresponds to the complete decomposition family, which has size $\Theta(n^3)$. Algorithm 8 with the complete decomposition family is equivalent to Algorithm 7. Fortunately, we can construct sparse decomposition families of size $O(n)$, which leads to a linear time approximate parsing algorithm.

Algorithm 8 Parsing with a decomposition family

```
Input: string $s$, decomposition family $(\mathcal{I}, \mathcal{D})$, context-free grammar $(\mathcal{X}, \mathcal{R})$
  for $X \in \mathcal{X}$ do
    for $i = 0$ to $n - 1$ do
      $COST[X, [i, i+1)] \leftarrow w(X \to s[i])$
    end for
  end for
  for $[i, j) \in \mathcal{I}$, ordered by increasing length do
    $COST[X, [i, j)] \leftarrow \infty$
    for $k$ such that $[i, j) \to [i, k)[k, j) \in \mathcal{D}$ do
      for $X \to YZ \in \mathcal{R}$ do
        $COST[X, [i, j)] \leftarrow \min\{COST[X, [i, j)],$
        $w(X \to YZ) + COST[Y, [i, k)] + COST[Z, [k, j)]\}$
      end for
    end for
  end for
```



s[0] s[1] s[2] s[3] s[4] s[5] s[6] s[7] s[8] s[9] s[10] s[11] s[12] s[13] s[14] s[15] s[16] s[17] s[18] s[19] s[20] s[21] s[22] s[23] s[24] s[25] s[26] s[27] s[28] s[29] s[30] s[31]
s[0] s[1] s[2] s[3] s[4] s[5] s[6] s[7] s[8] s[9] s[10] s[11] s[12] s[13] s[14] s[15] s[16] s[17] s[18] s[19] s[20] s[21] s[22] s[23] s[24] s[25] s[26] s[27] s[28] s[29] s[30] s[31]
s[0] s[1] s[2] s[3] s[4] s[5] s[6] s[7] s[8] s[9] s[10] s[11] s[12] s[13] s[14] s[15] s[16] s[17] s[18] s[19] s[20] s[21] s[22] s[23] s[24] s[25] s[26] s[27] s[28] s[29] s[30] s[31]
s[0] s[1] s[2] s[3] s[4] s[5] s[6] s[7] s[8] s[9] s[10] s[11] s[12] s[13] s[14] s[15] s[16] s[17] s[18] s[19] s[20] s[21] s[22] s[23] s[24] s[25] s[26] s[27] s[28] s[29] s[30] s[31]

Figure 6.2: Illustrating the construction from Theorem 6.4.1, with $k = 4$. Rectangles denote portions of the string between members of the index set. A selection of intervals that live at each level are shown.

6.4 Constructing Sparse Decomposition Families

Theorem 6.4.1. *Let s be a string of length n, and let k be an integer. Then there is a $\frac{1}{k}$-flexible decomposition family $(\mathcal{I}, \mathcal{D})$ for s that has at most $2nk^2$ decompositions.*

Proof. Let C be the set of integers from 0 to n, inclusive. We will call this the index set. We recursively define a sequence of subsampled index sets $C^{(i)}$ as follows: $C^{(0)} = C$, and $C^{(i+1)}$ is obtained from $C^{(i)}$ by taking every other index. The number of indices in all $C^{(i)}$ combined is at most $2n$.

We will talk about the length of an interval relative to an index set. An interval $[a, b)$ has length ℓ in index set $C^{(i)}$ if there are $\ell - 1$ indices between a and b in $C^{(i)}$.

An interval $[a, b)$ is *allowable* if there is some $C^{(i)}$ which contains both a and b, and $[a, b)$

has length at most k in $C^{(i)}$, i.e., there are at most $k-1$ other indices between them in $C^{(i)}$. The number of allowable intervals is then at most $2nk$.

We will say that an interval $[a, b)$ *lives* in level i (for $i \geq 0$) if a and b are both in $C^{(i)}$, and if the length of the interval relative to $C^{(i)}$ is strictly greater than $k/2$ and at most k. When $i = 0$, we will not have the minimum length requirement, so all intervals of C of length at most k live in level 0. It is straightforward to see that each allowable interval lives in a single level.

If $[a, b)$ is an interval that lives in level i, then we can decompose it as $[a, b) \to [a, c)[c, b)$ by picking midpoints c from among the indices in $C^{(i)}$. Some of these choices will lead to intervals that live in level $i - 1$.

This decomposition family is $\frac{1}{k}$-flexible. When picking midpoints at level i, our interval has length at least $k/2$ in $C^{(i)}$, so we have at least $k/2$ evenly spaced midpoint choices. We can thus choose a midpoint that is within c points of any desired midpoint, where c is at most $\frac{1}{k}$ times the length of the interval we are splitting.

There are at most k midpoints per allowable interval, so the number of composition rules is at most $2nk^2$. □

This demonstrates the existence of linear-size θ-flexible decomposition families, which yields a linear time approximate parsing algorithm. This in turn means that we can use context-free grammars to parse long strings, which expands the range of domains where we can use context-free grammar models.

APPENDIX A

MODELS OF LEAVES

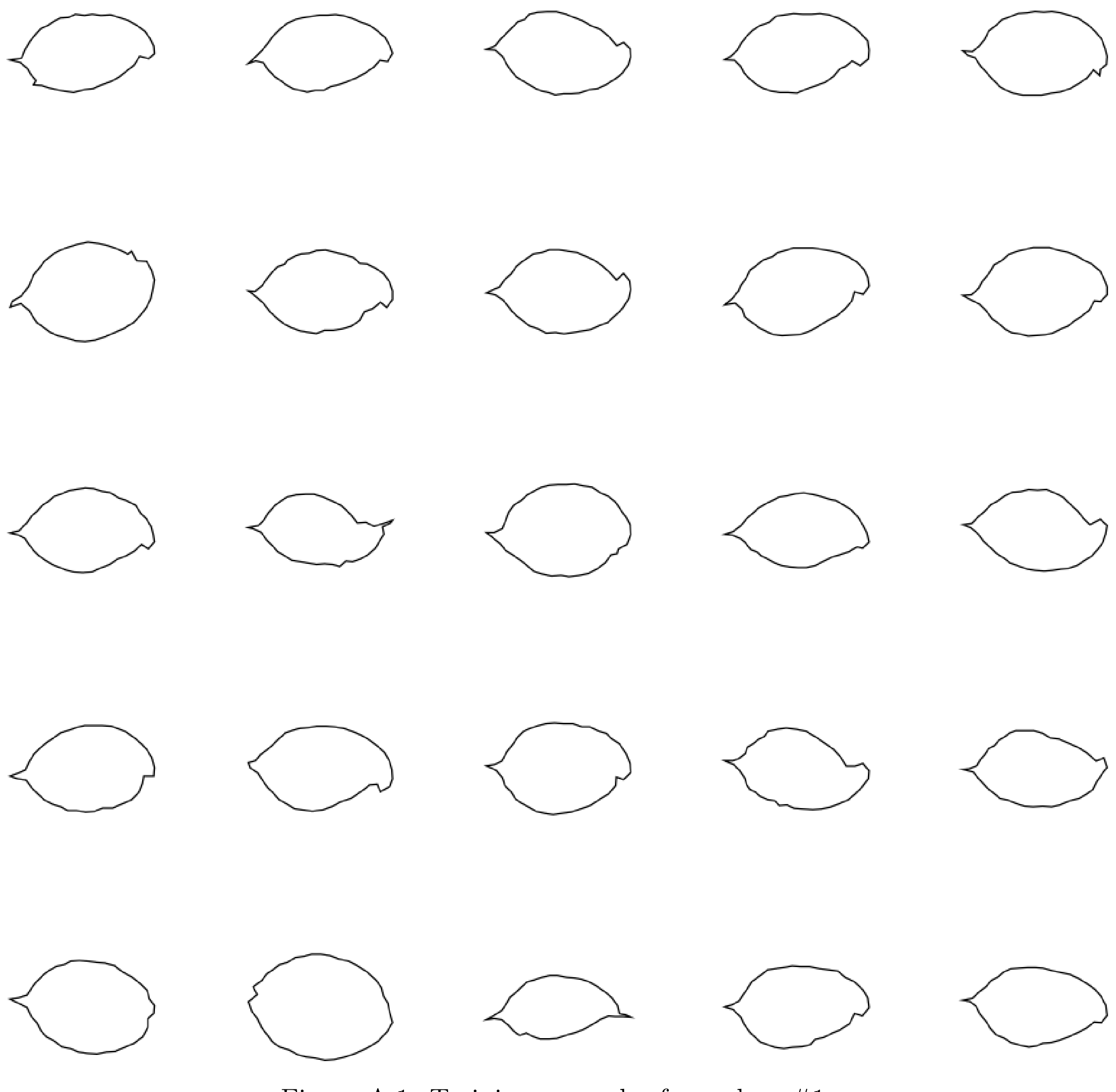


Figure A.1: Training examples from class #1.

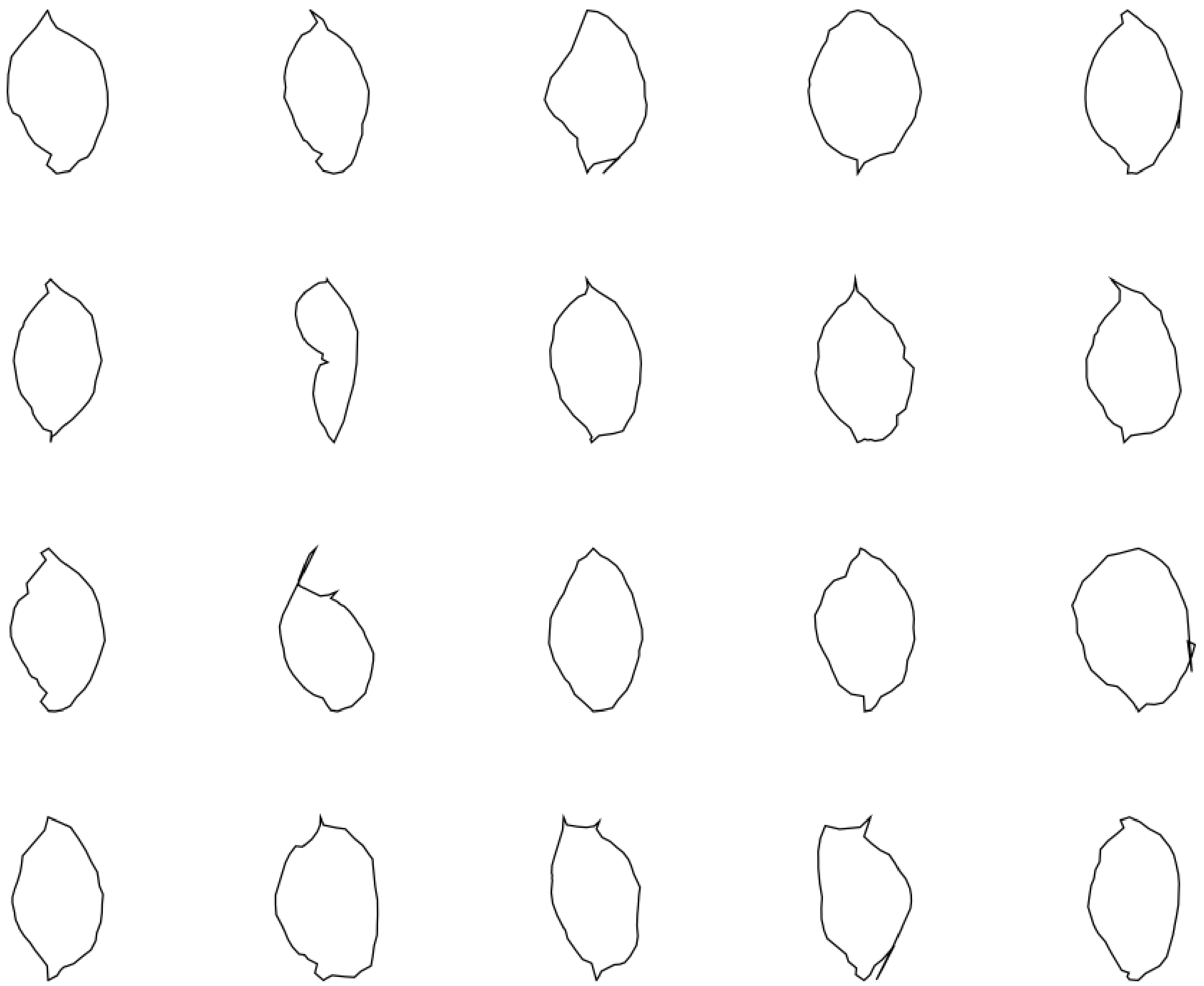


Figure A.2: Samples from learned model for class #1.

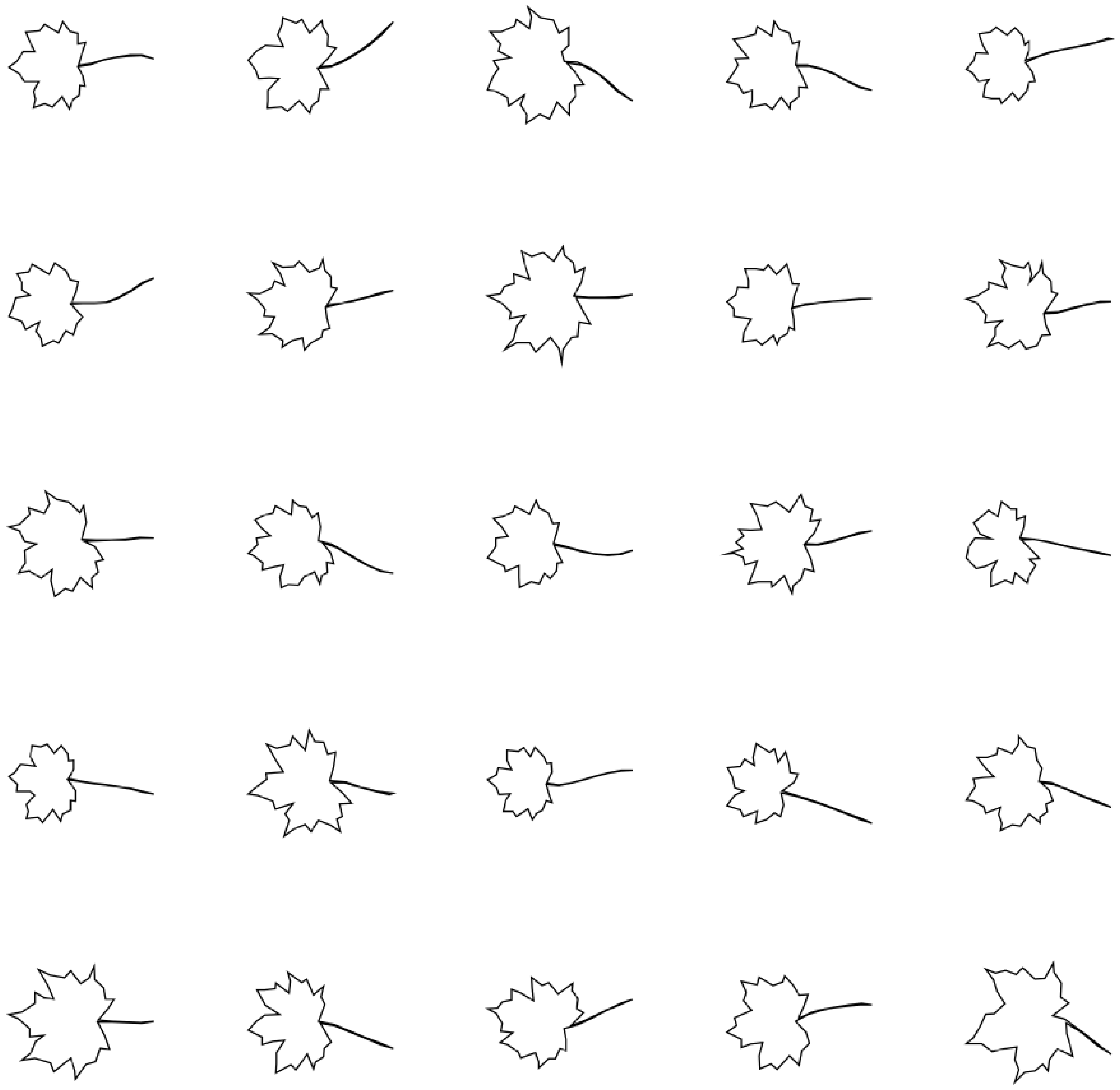


Figure A.3: Training examples from class #2.

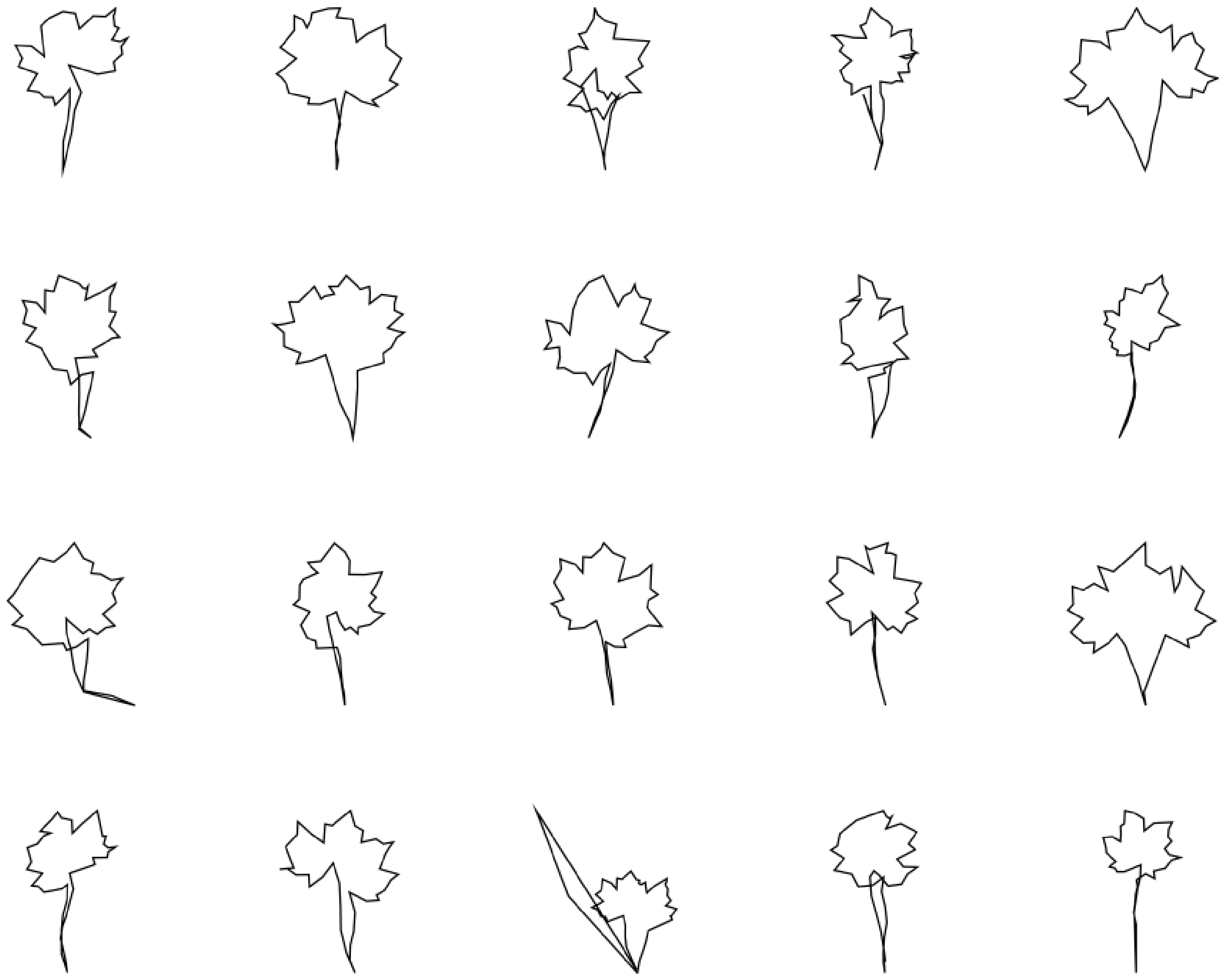


Figure A.4: Samples from learned model for class #2.

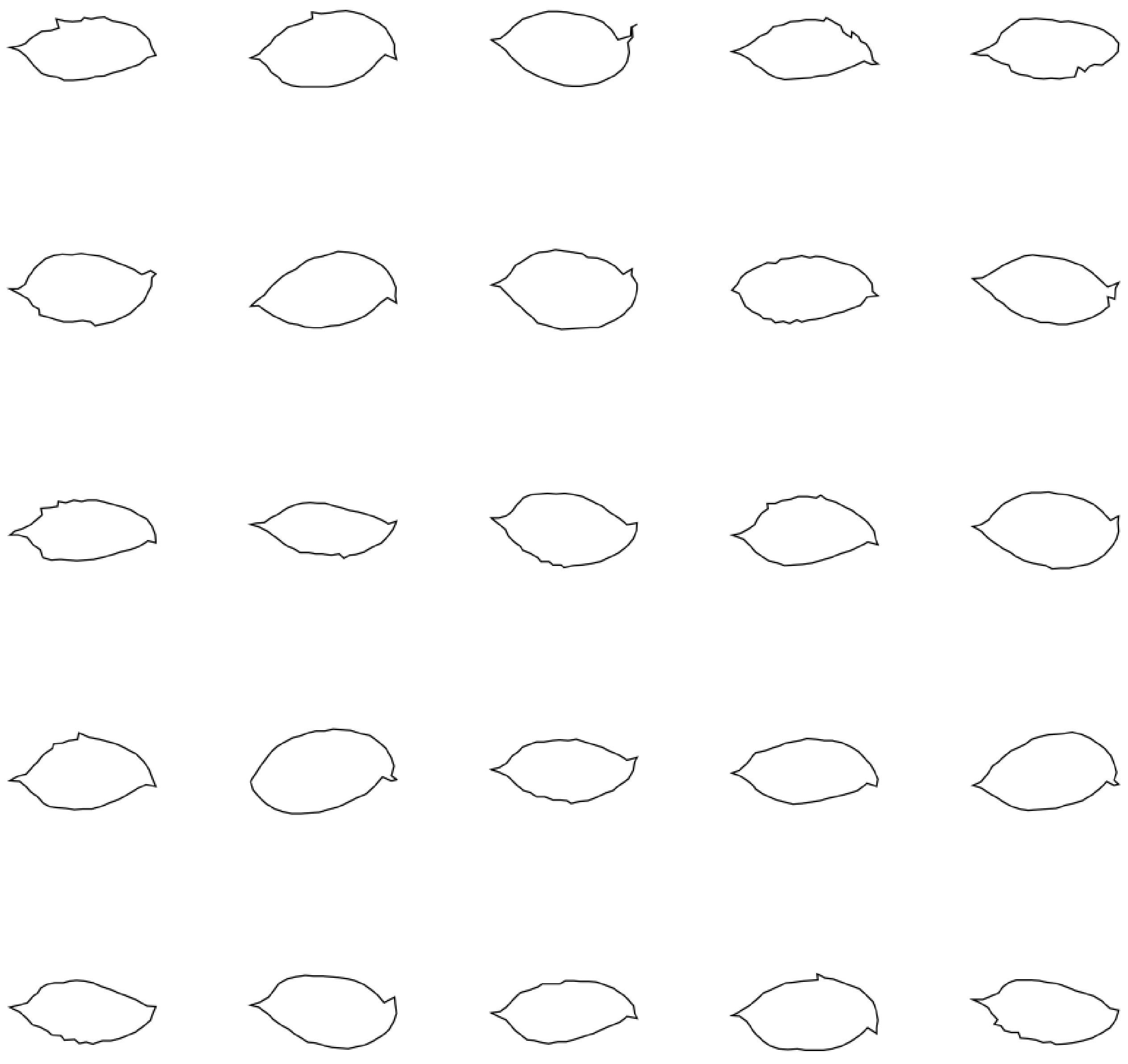


Figure A.5: Training examples from class #3.

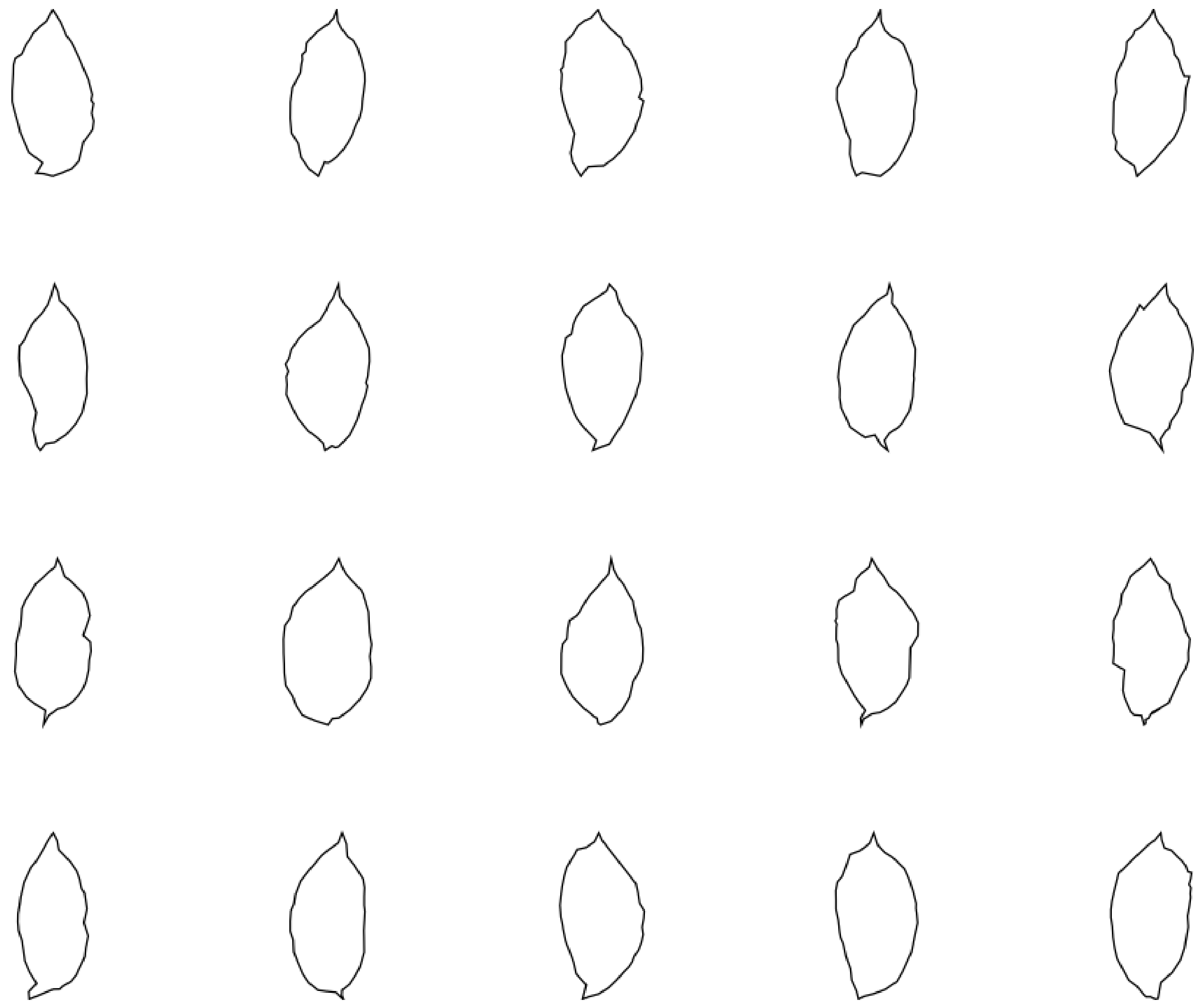


Figure A.6: Samples from learned model for class #3.

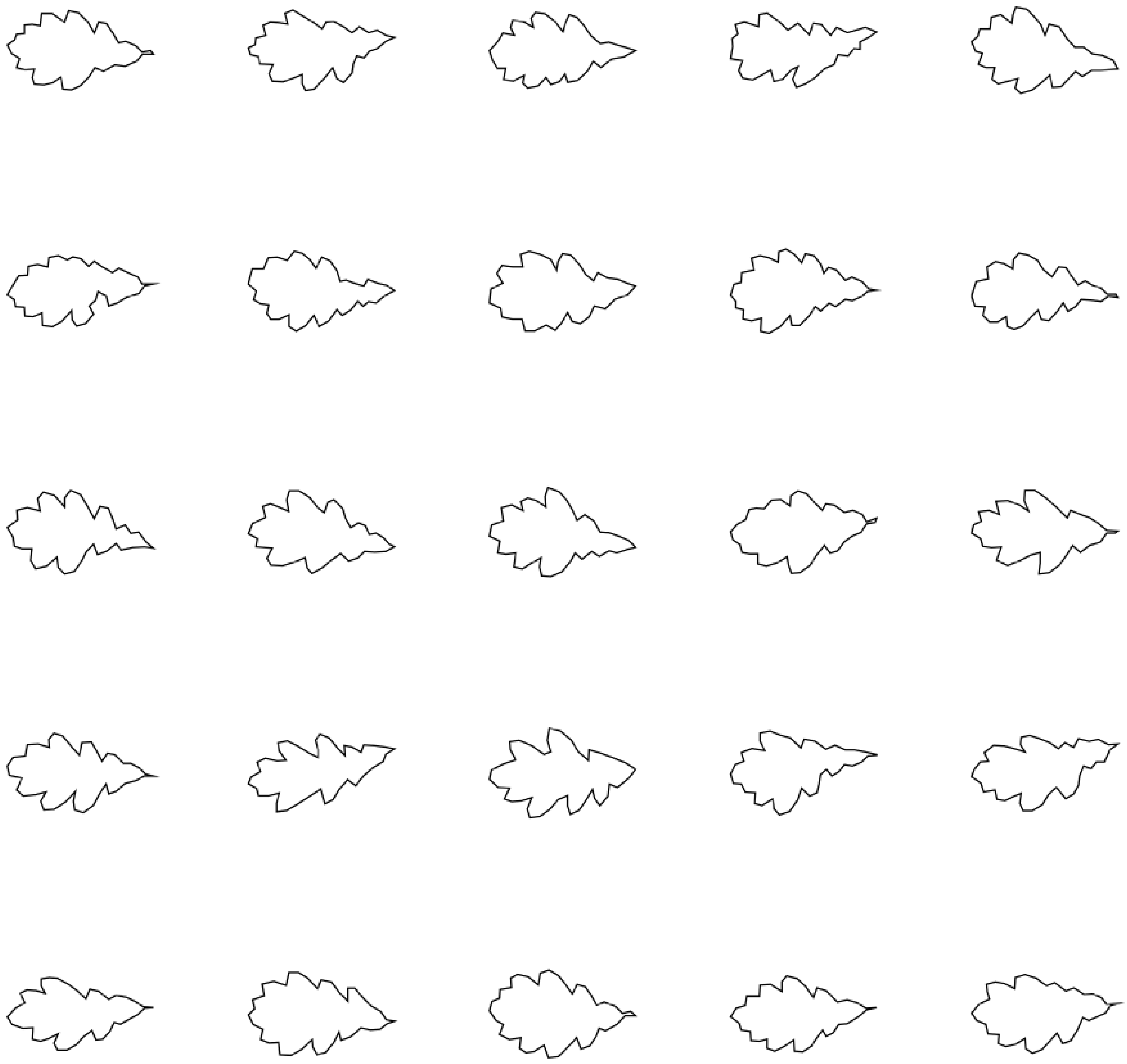


Figure A.7: Training examples from class #4.

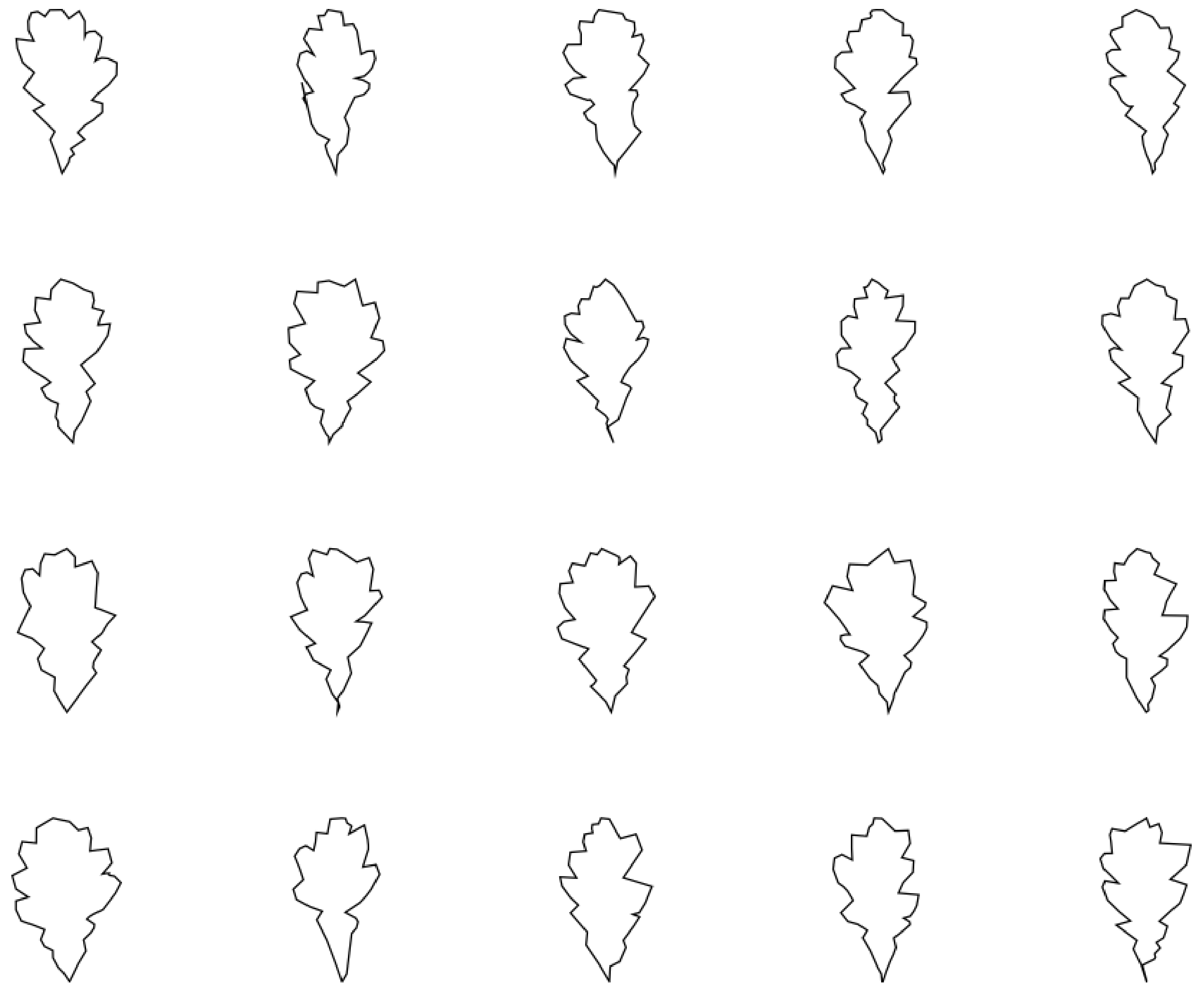


Figure A.8: Samples from learned model for class #4.

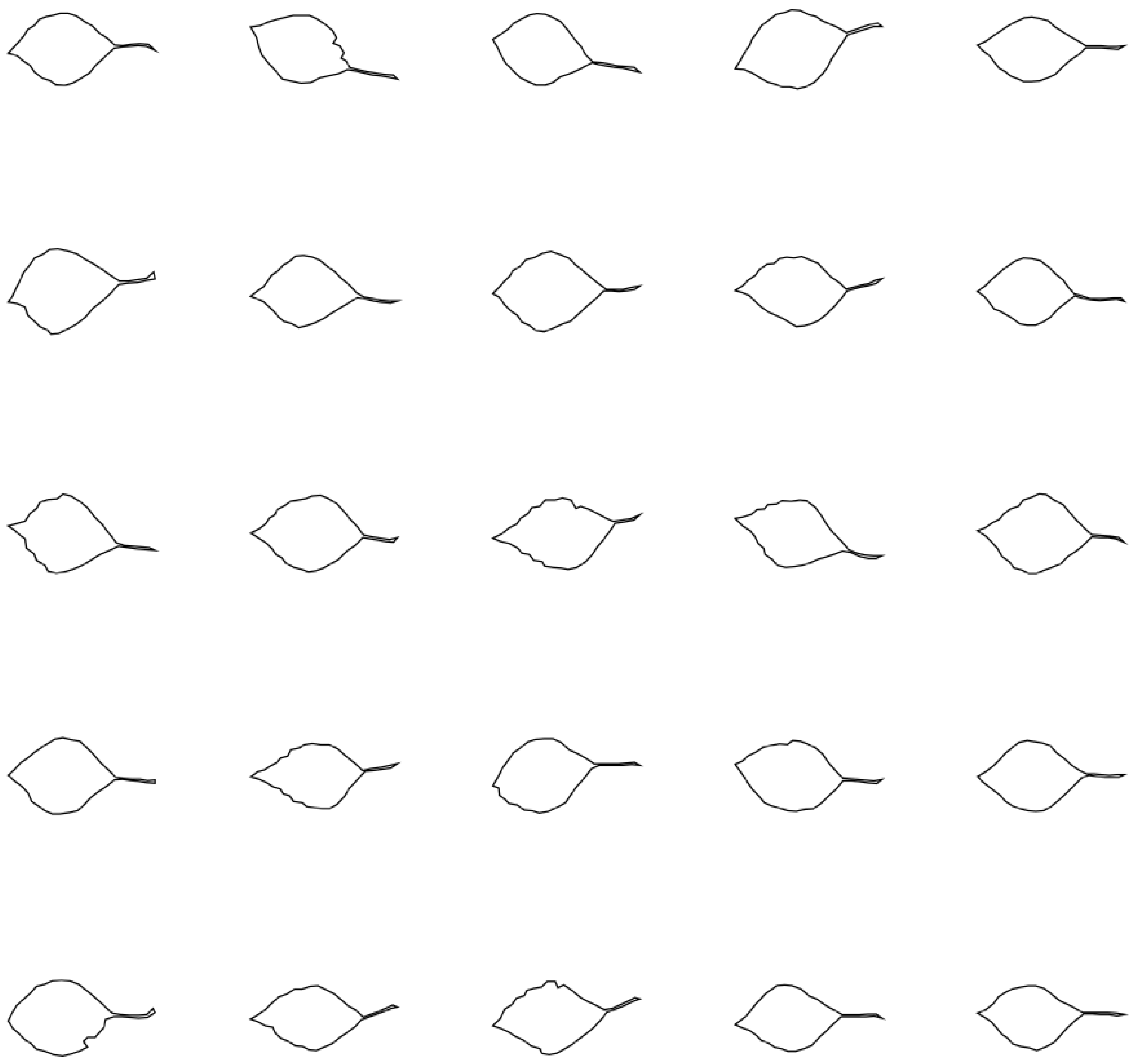


Figure A.9: Training examples from class #5.

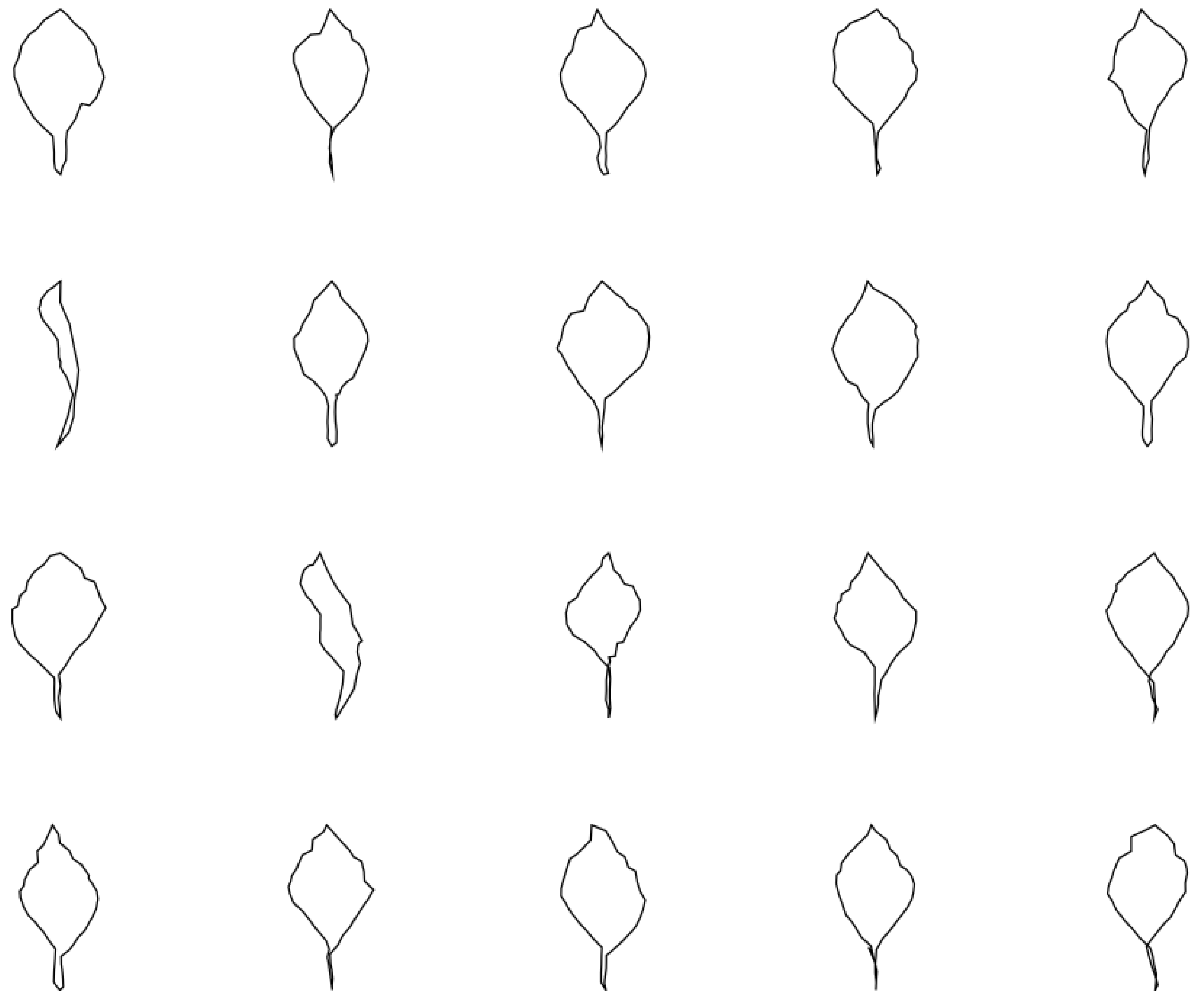


Figure A.10: Samples from learned model for class #5.

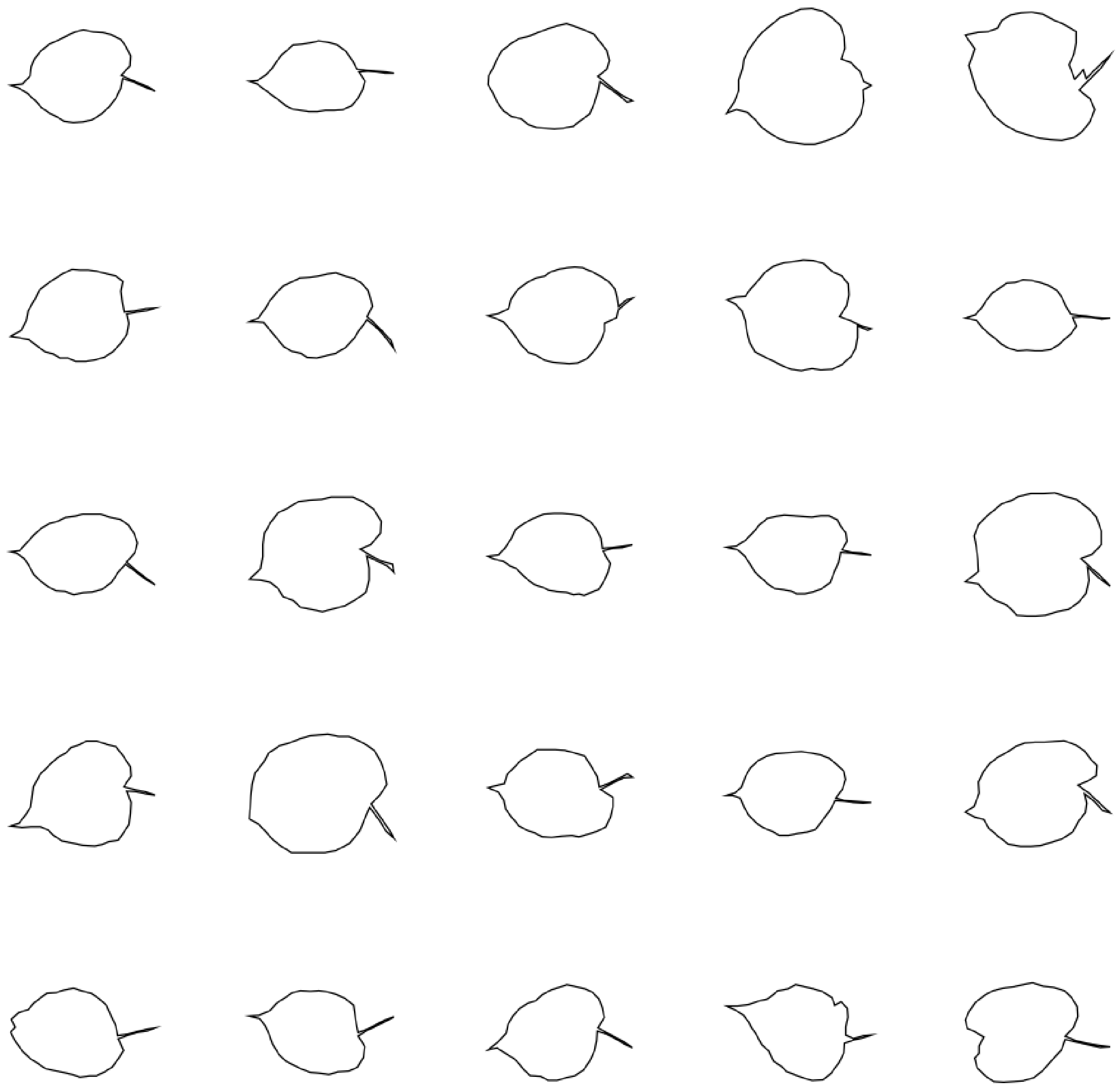


Figure A.11: Training examples from class #6.

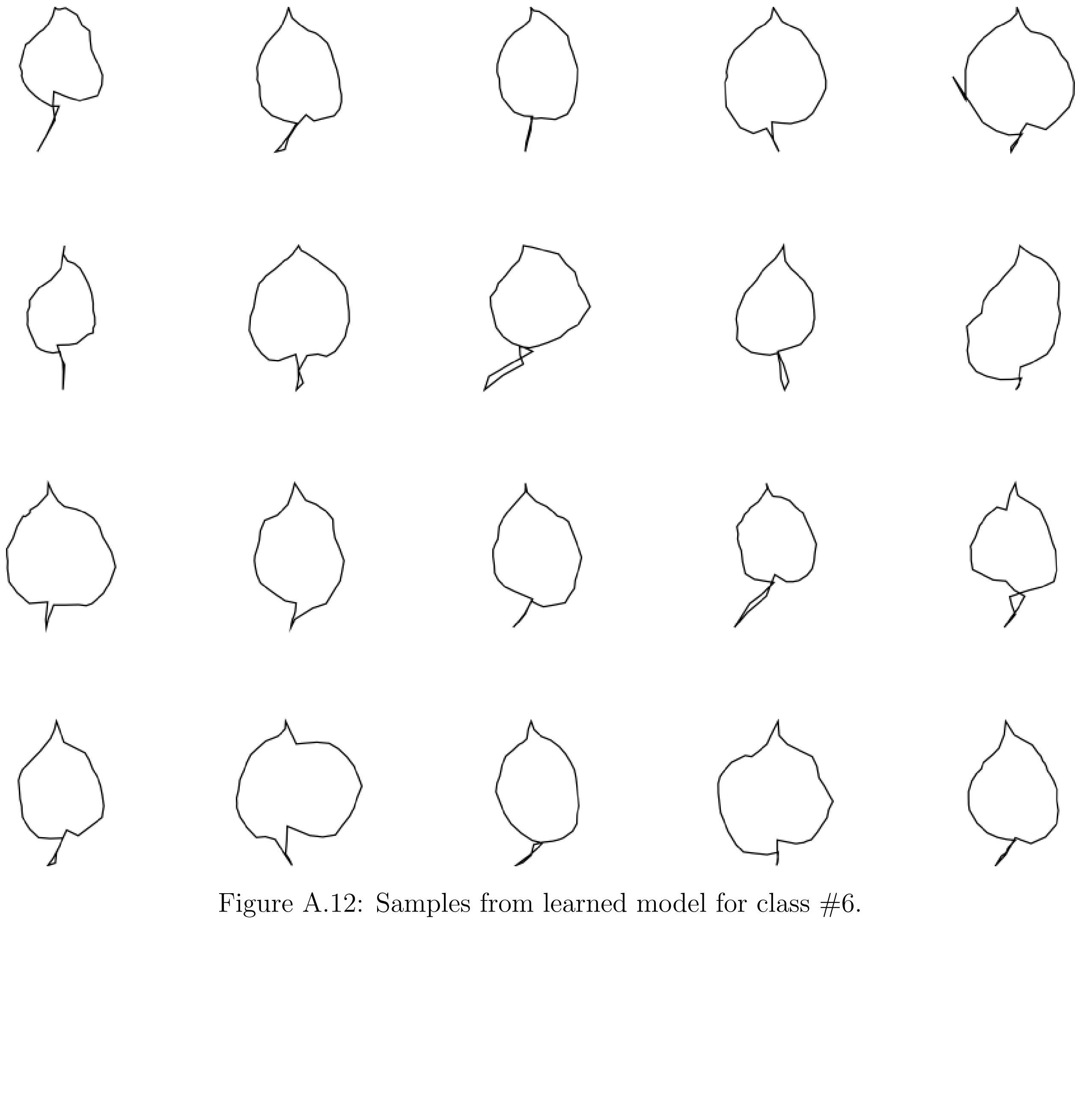


Figure A.12: Samples from learned model for class #6.

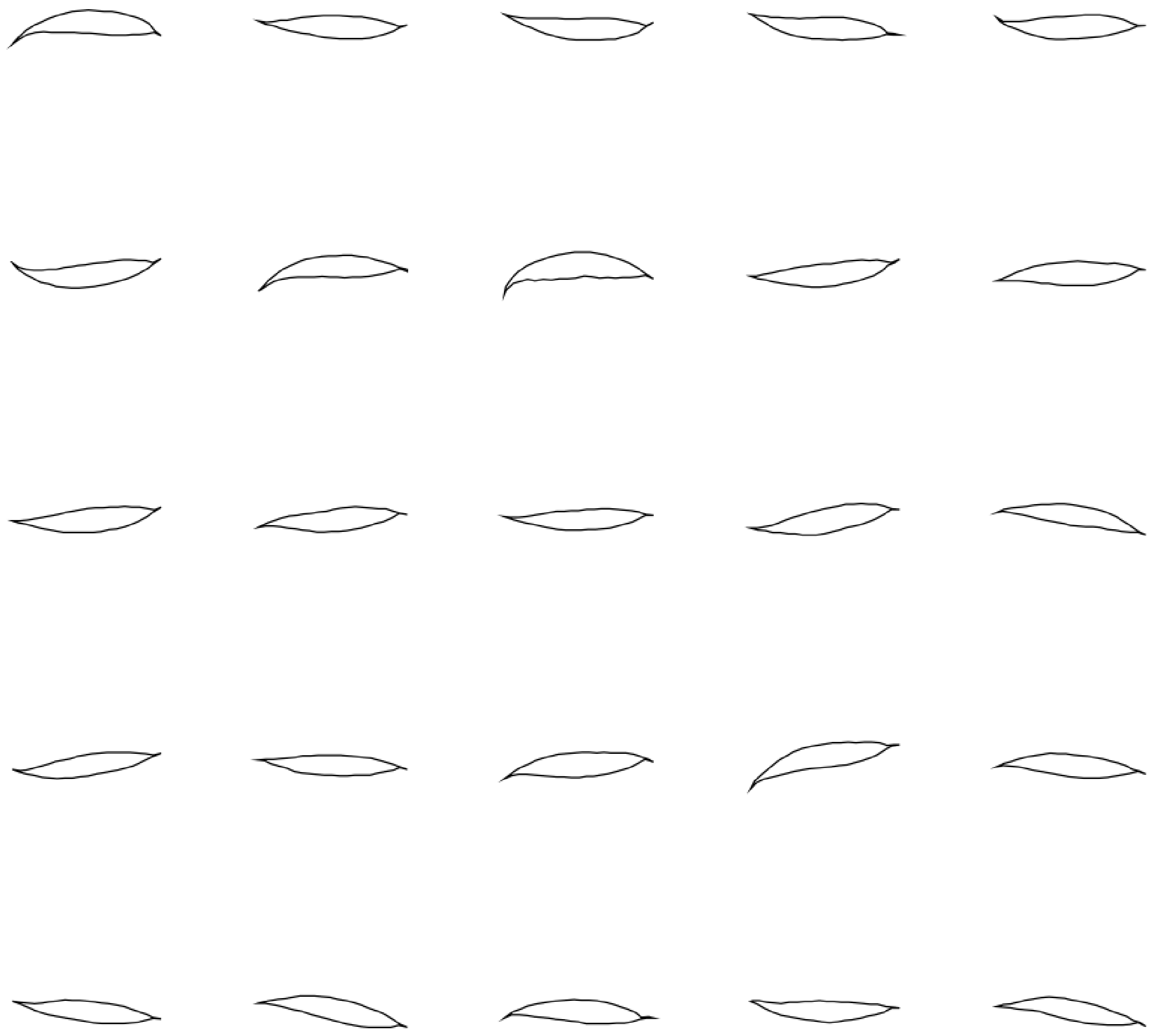


Figure A.13: Training examples from class #7.

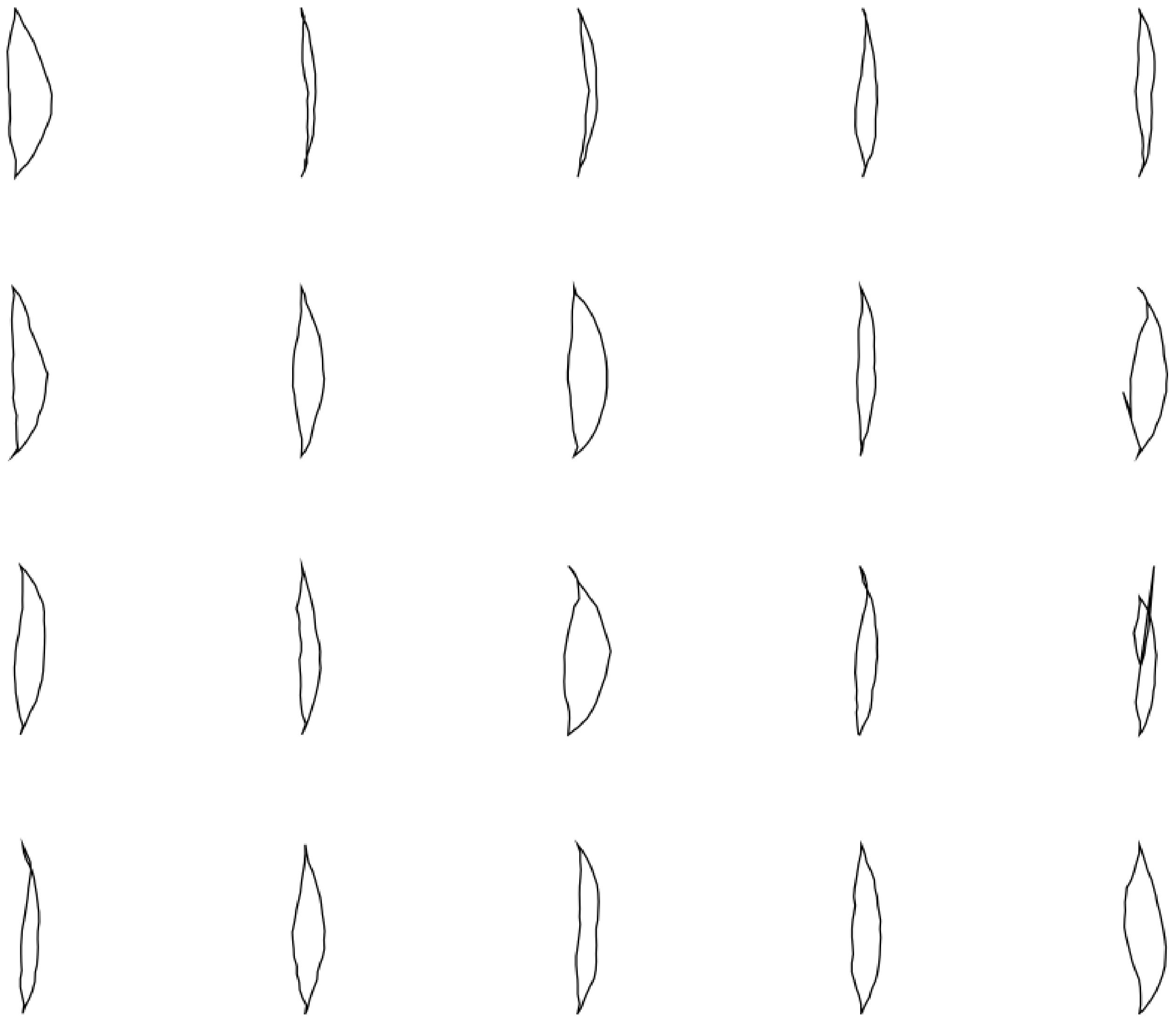


Figure A.14: Samples from learned model for class #7.

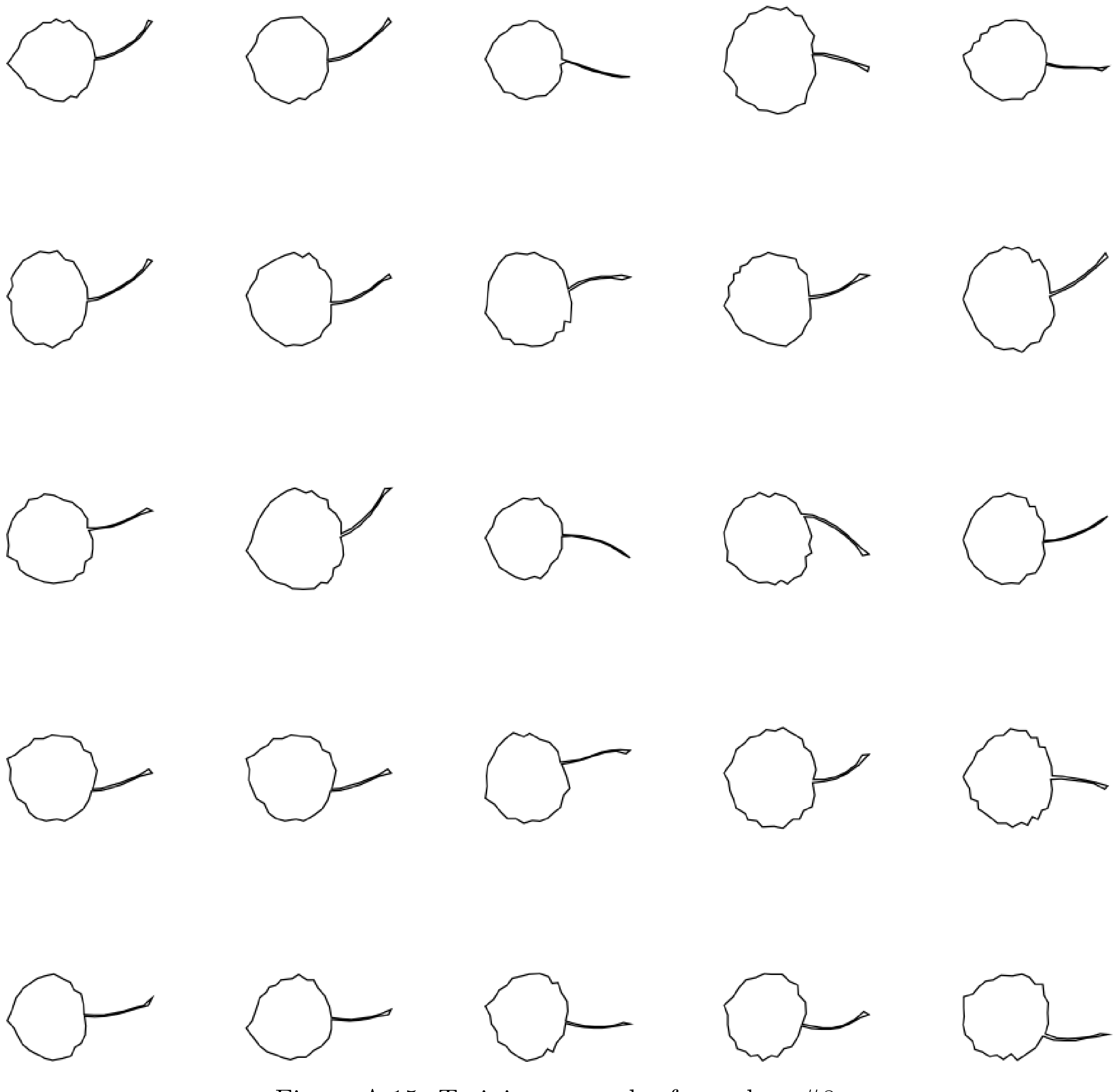


Figure A.15: Training examples from class #8.

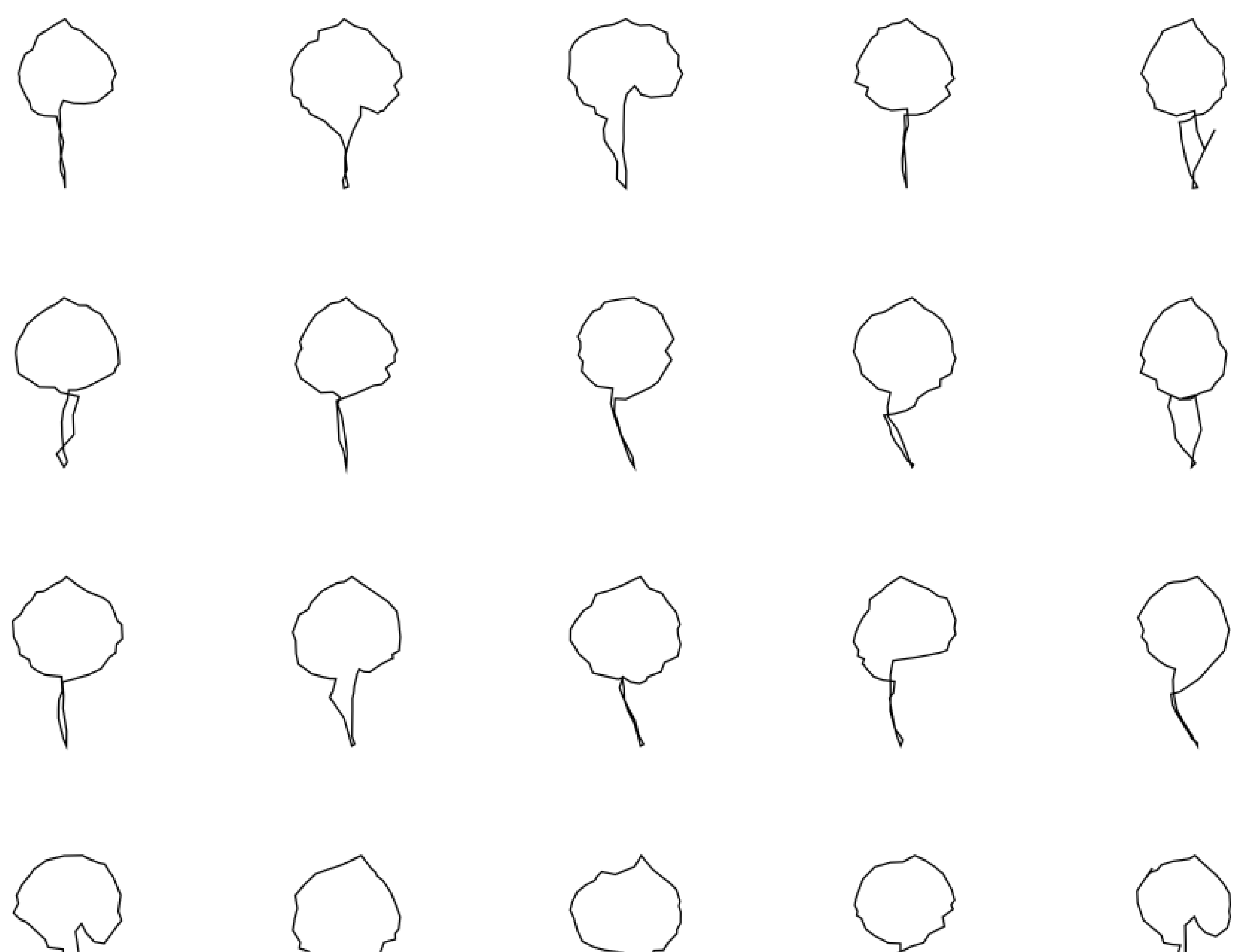


Figure A.16: Samples from learned model for class #8.

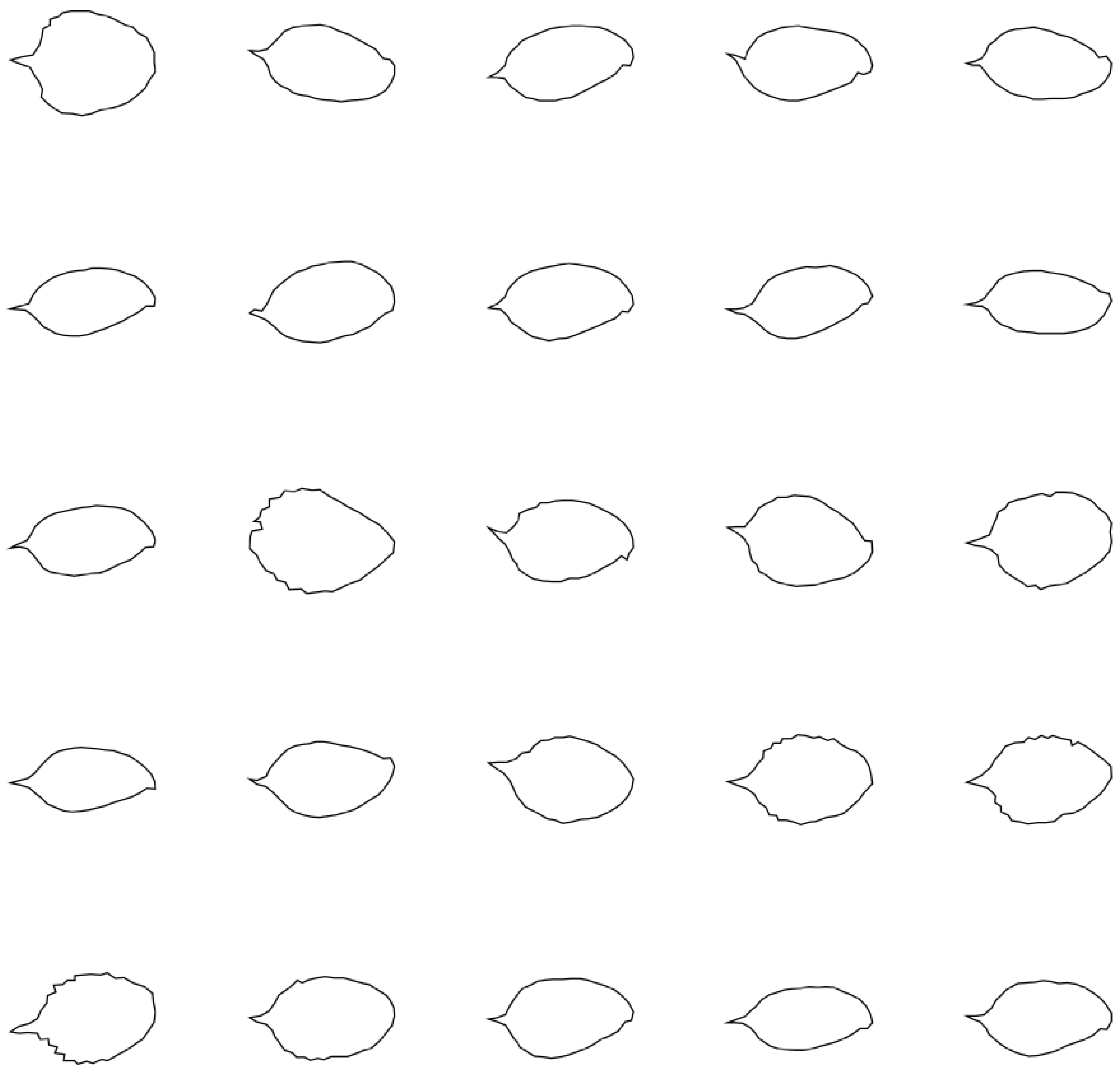


Figure A.17: Training examples from class #9.

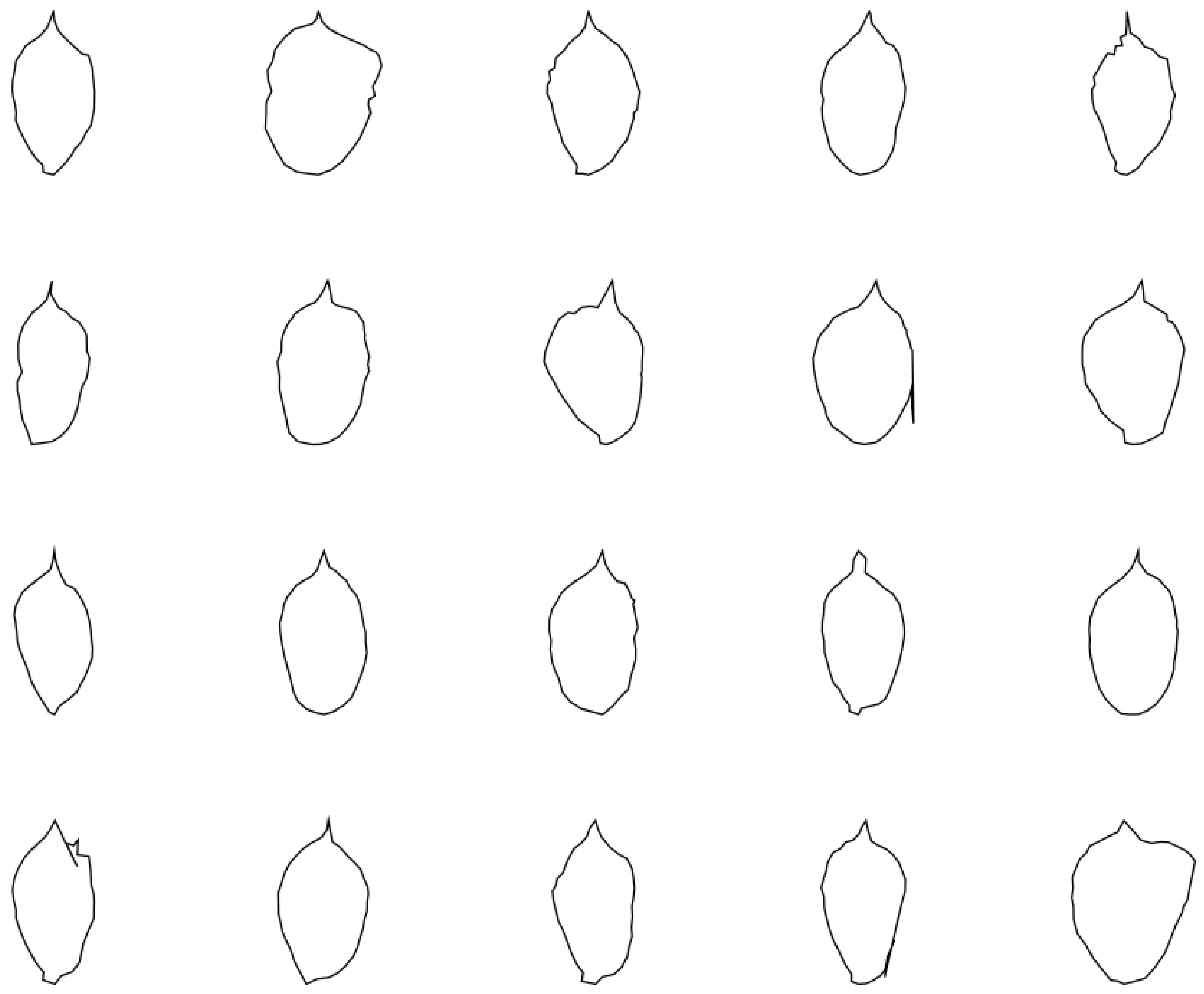


Figure A.18: Samples from learned model for class #9.

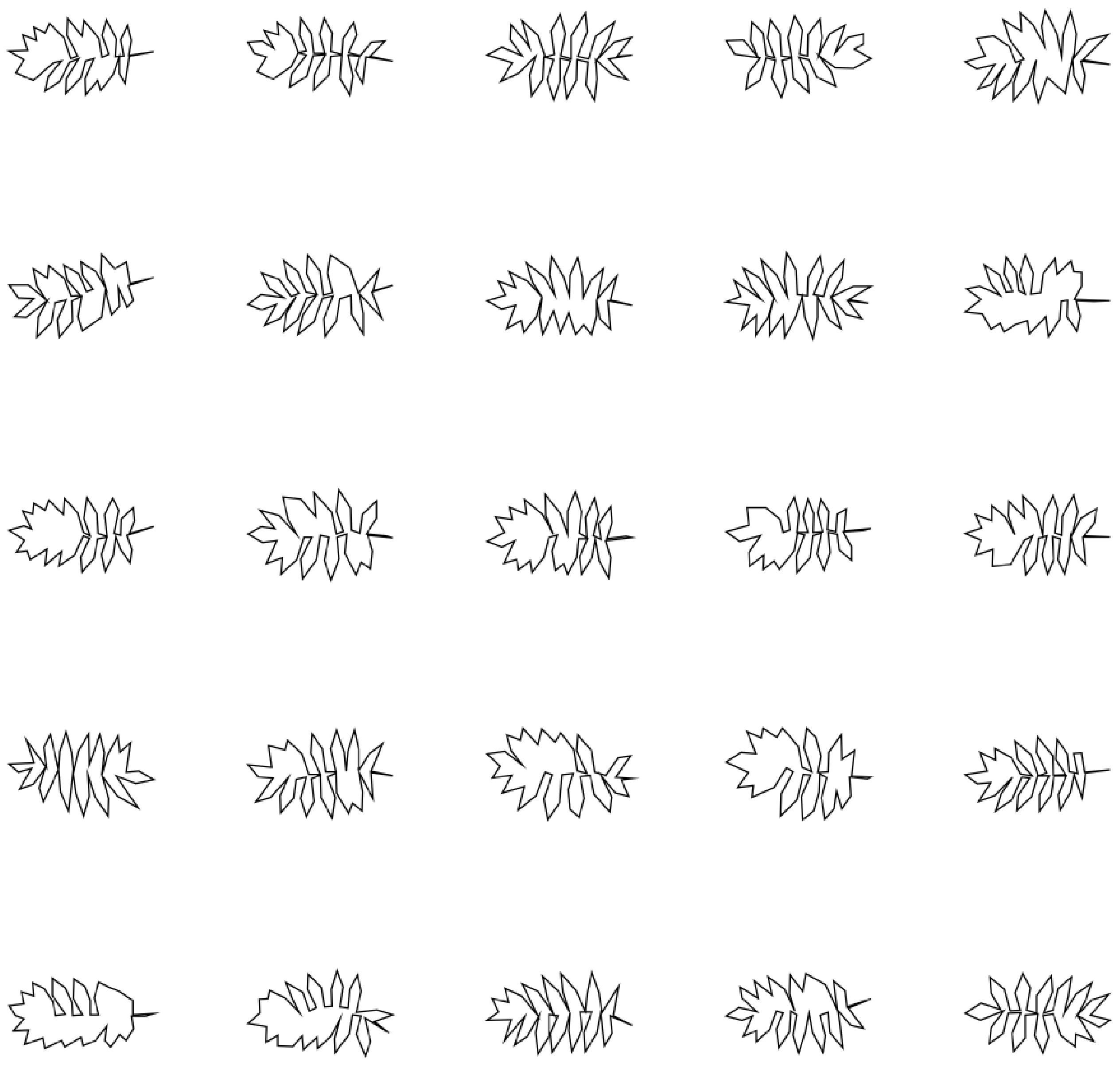


Figure A.19: Training examples from class #10.

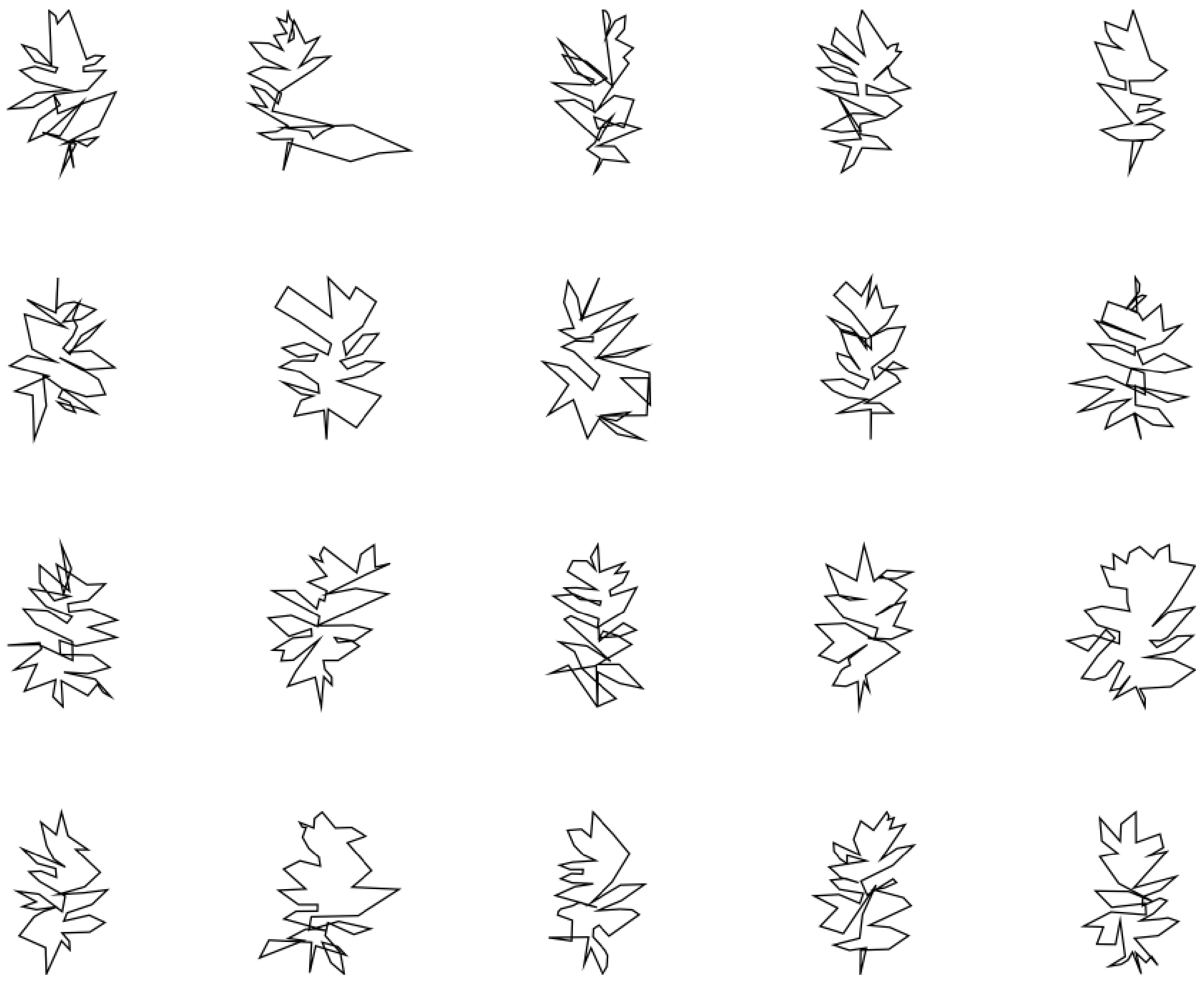


Figure A.20: Samples from learned model for class #10.

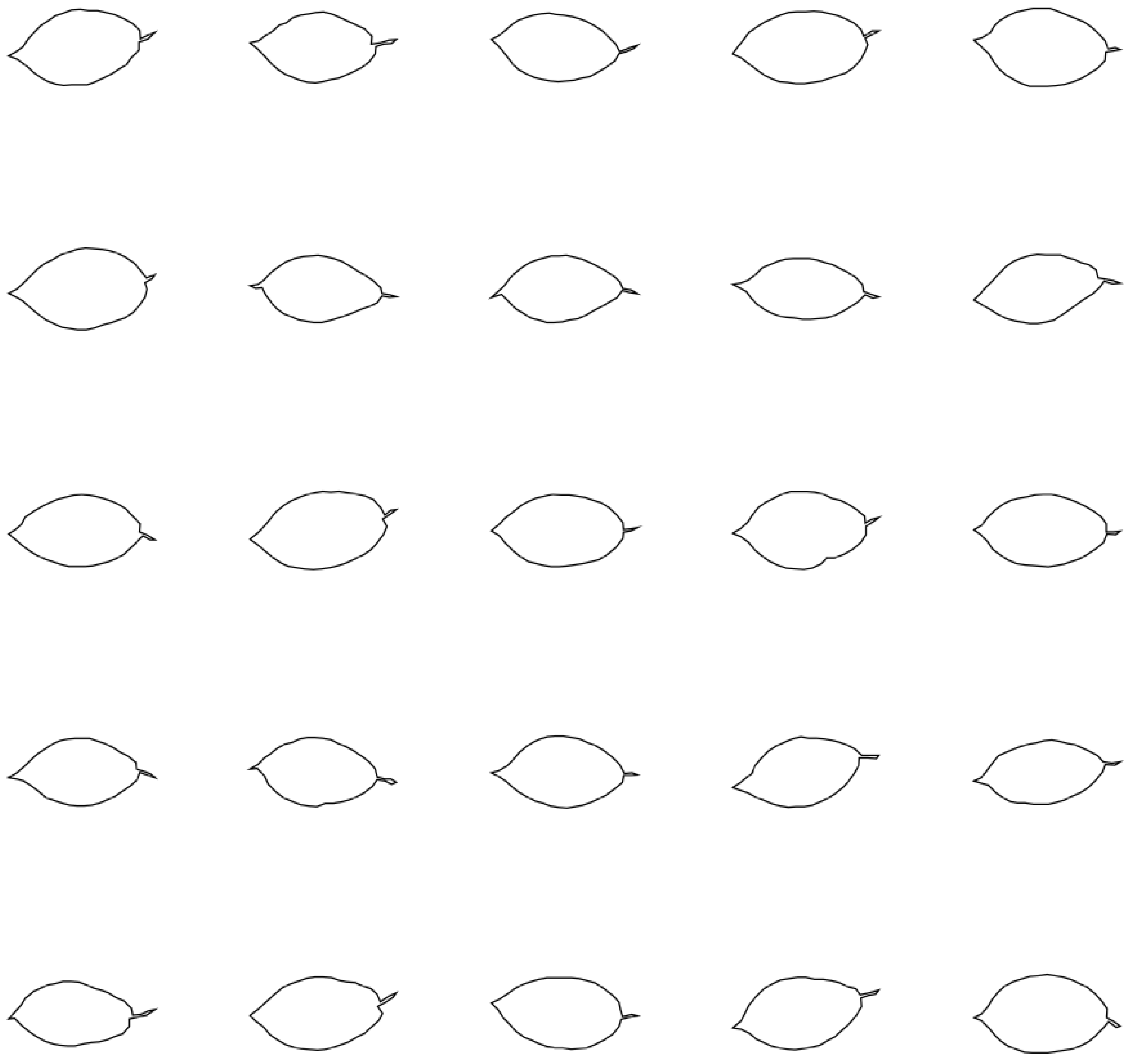


Figure A.21: Training examples from class #11.

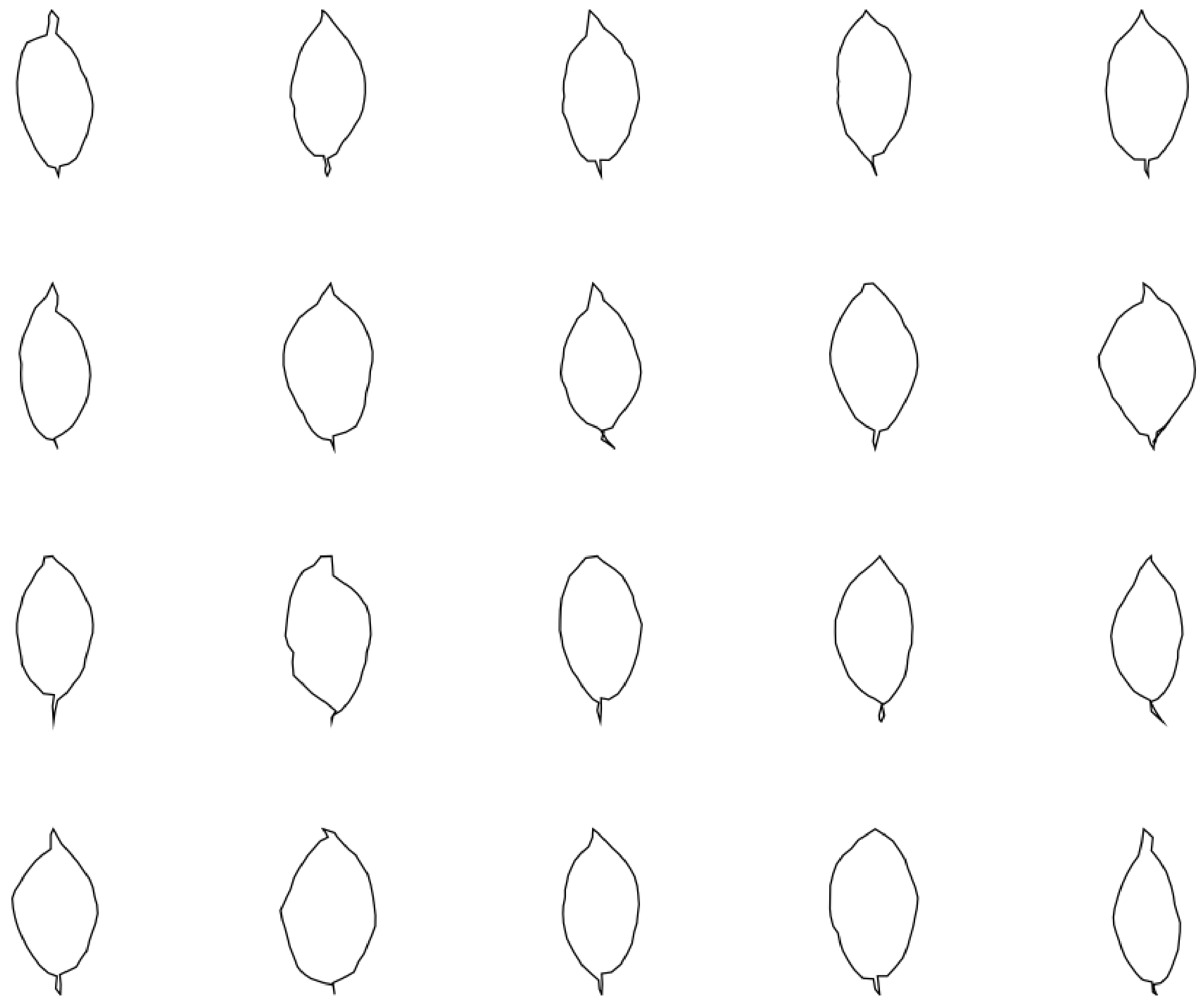


Figure A.22: Samples from learned model for class #11.

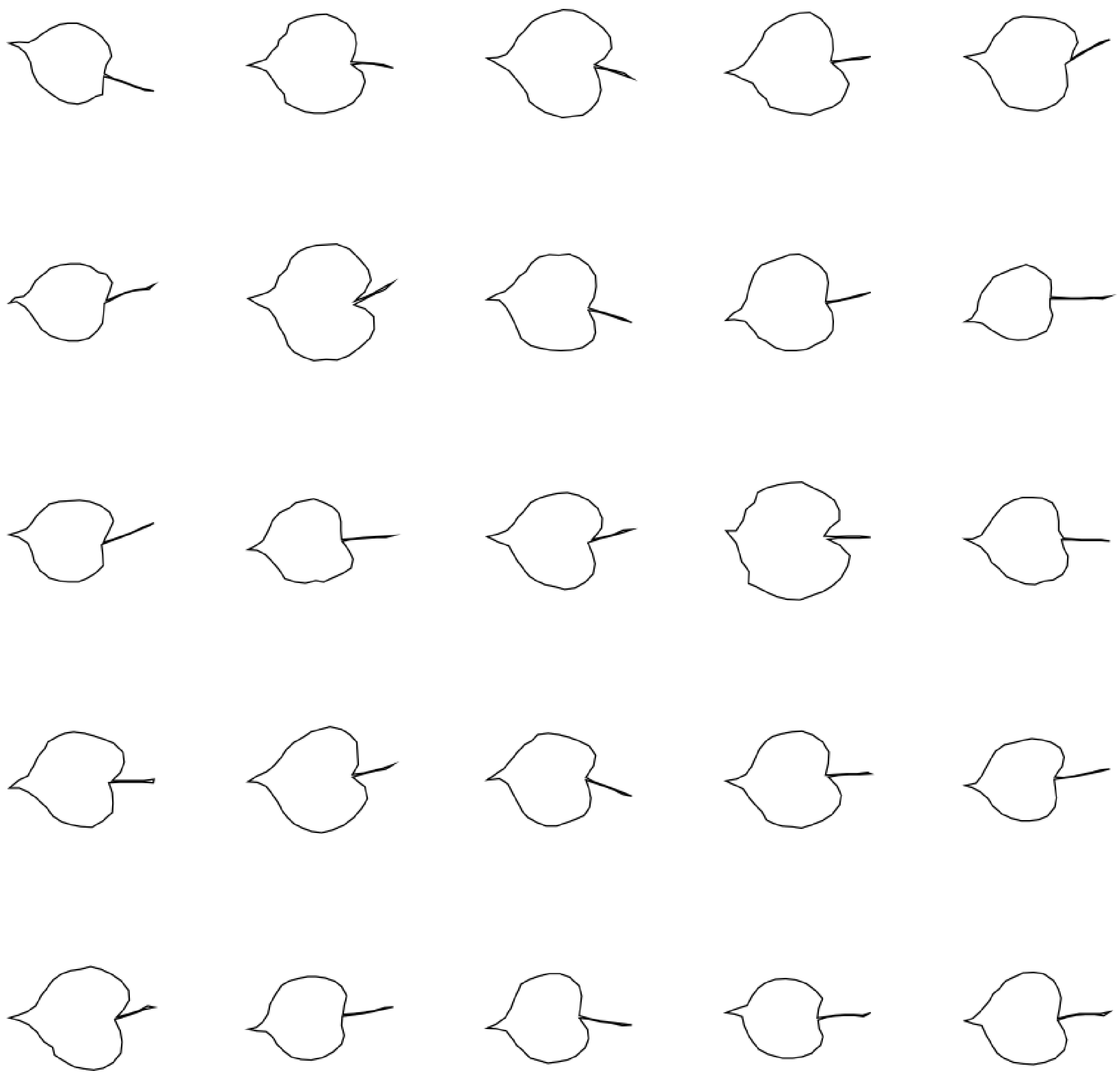


Figure A.23: Training examples from class #12.

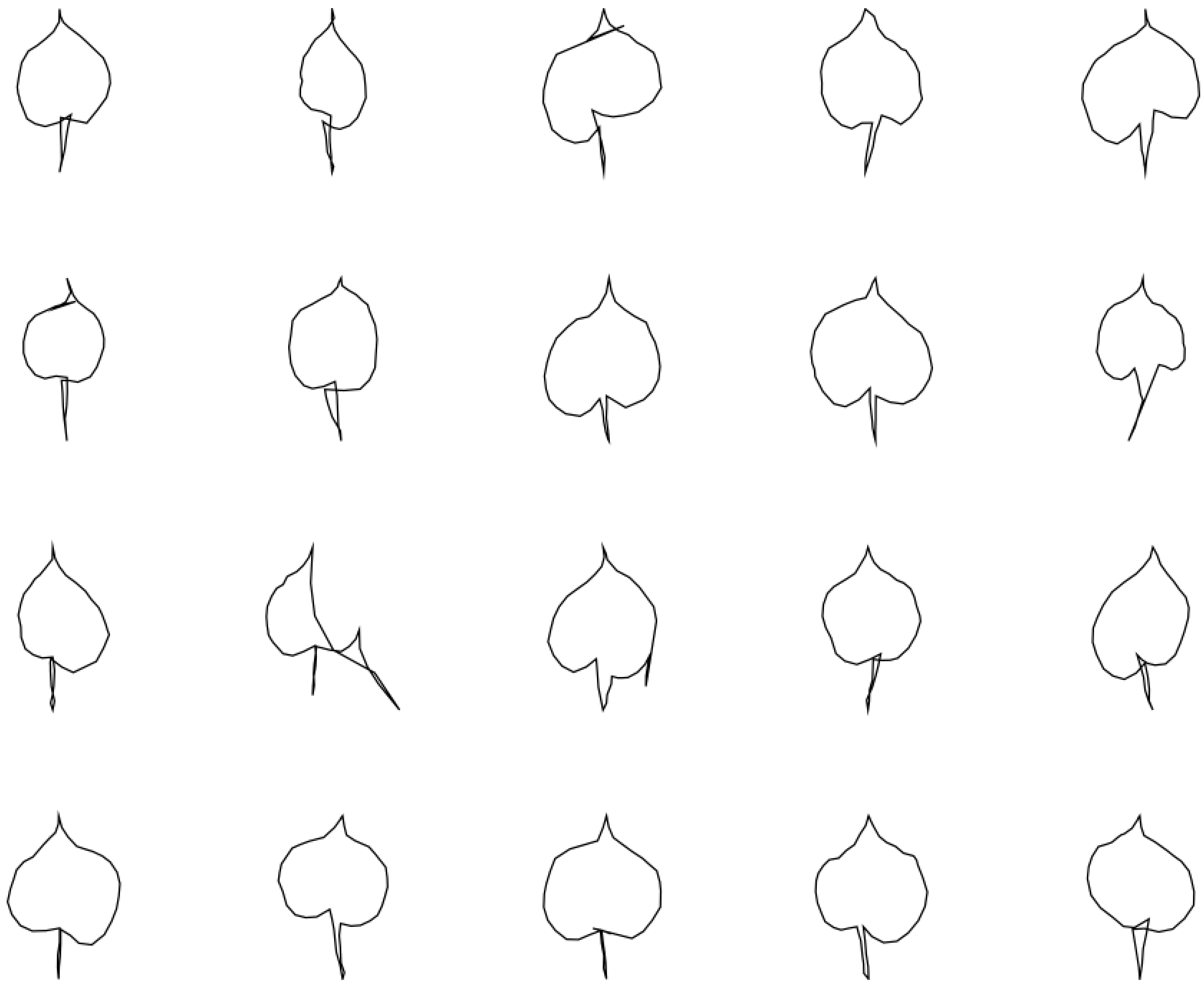


Figure A.24: Samples from learned model for class #12.

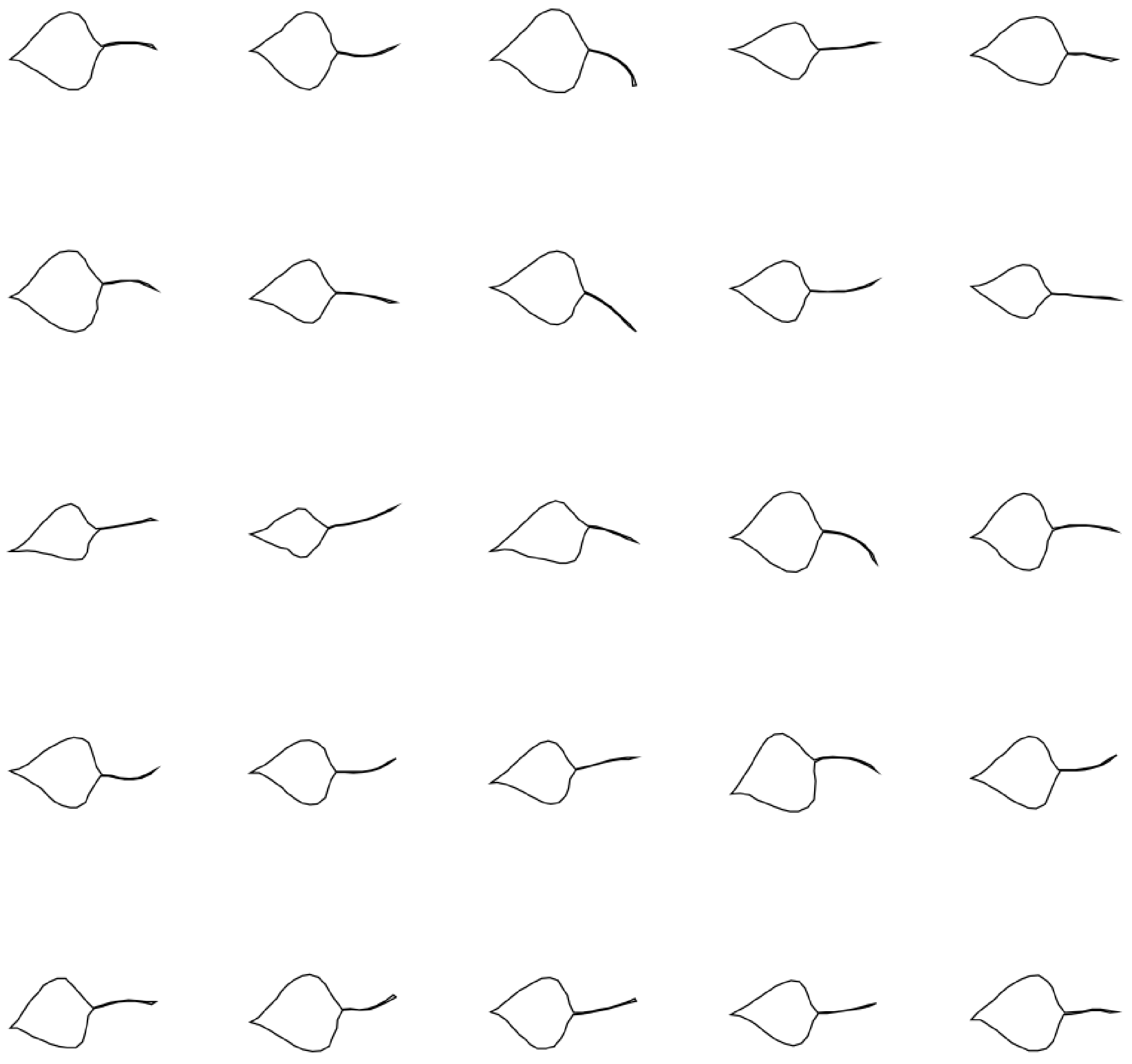


Figure A.25: Training examples from class #13.

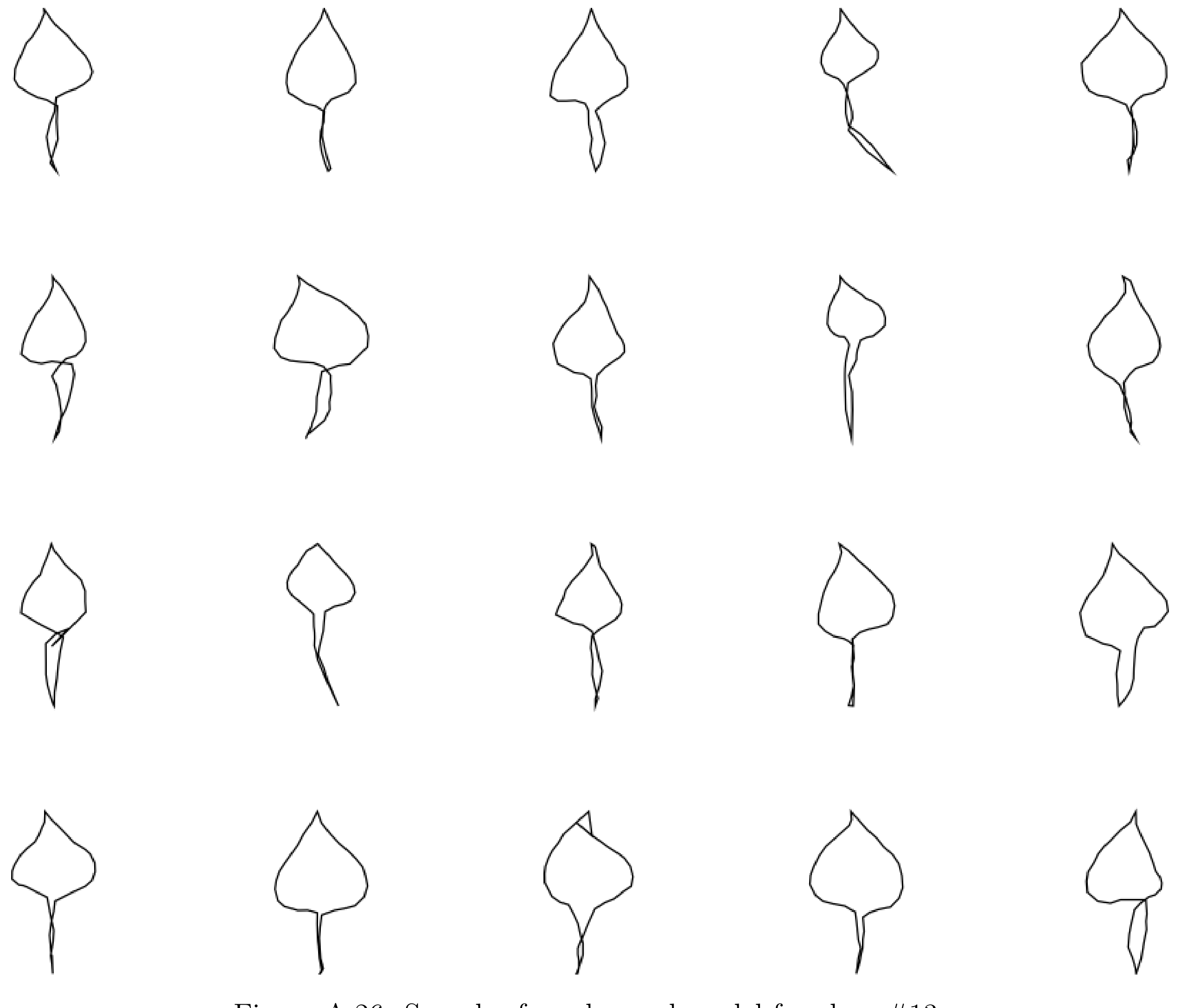


Figure A.26: Samples from learned model for class #13.

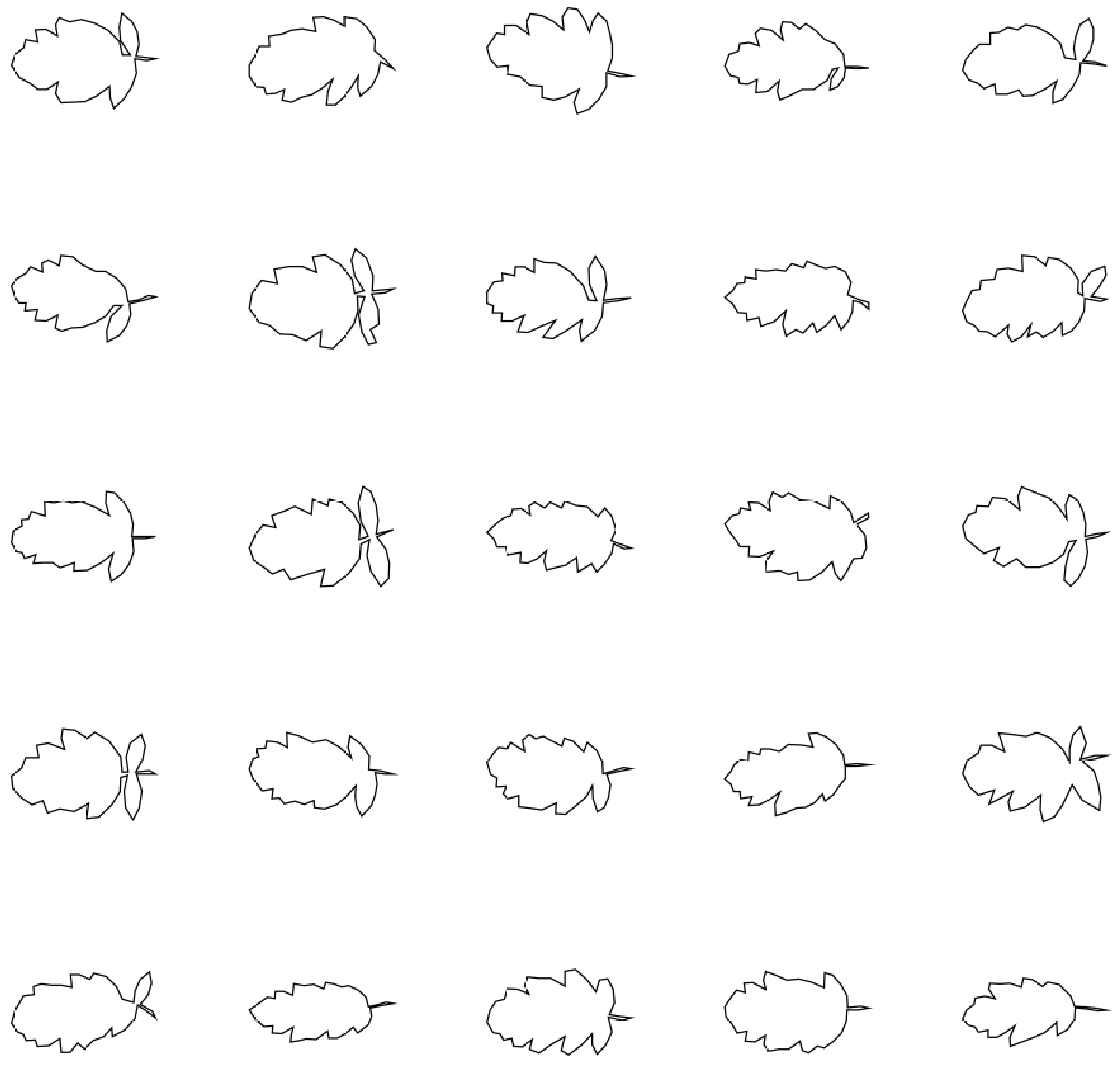


Figure A.27: Training examples from class #14.

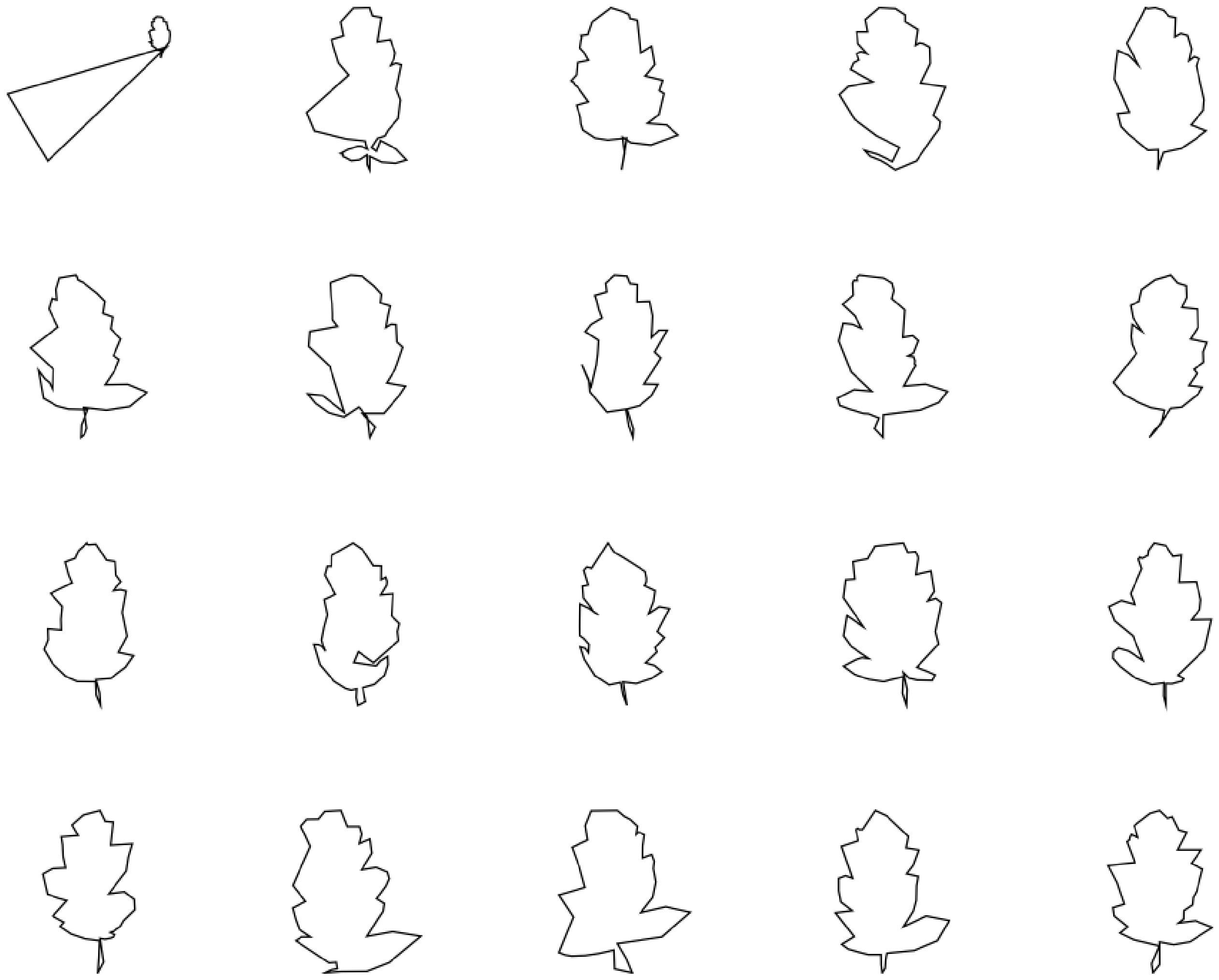


Figure A.28: Samples from learned model for class #14.

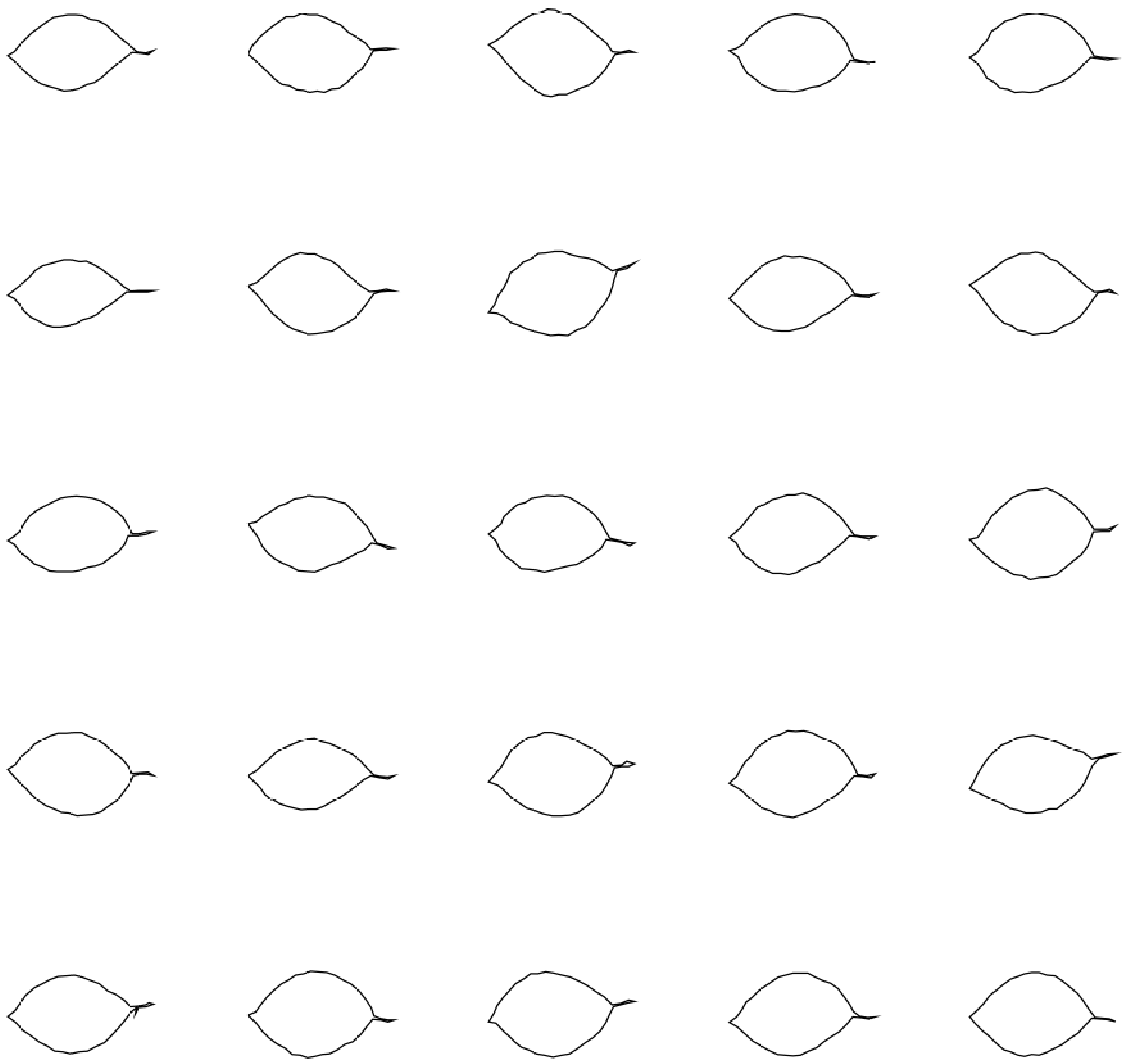


Figure A.29: Training examples from class #15.

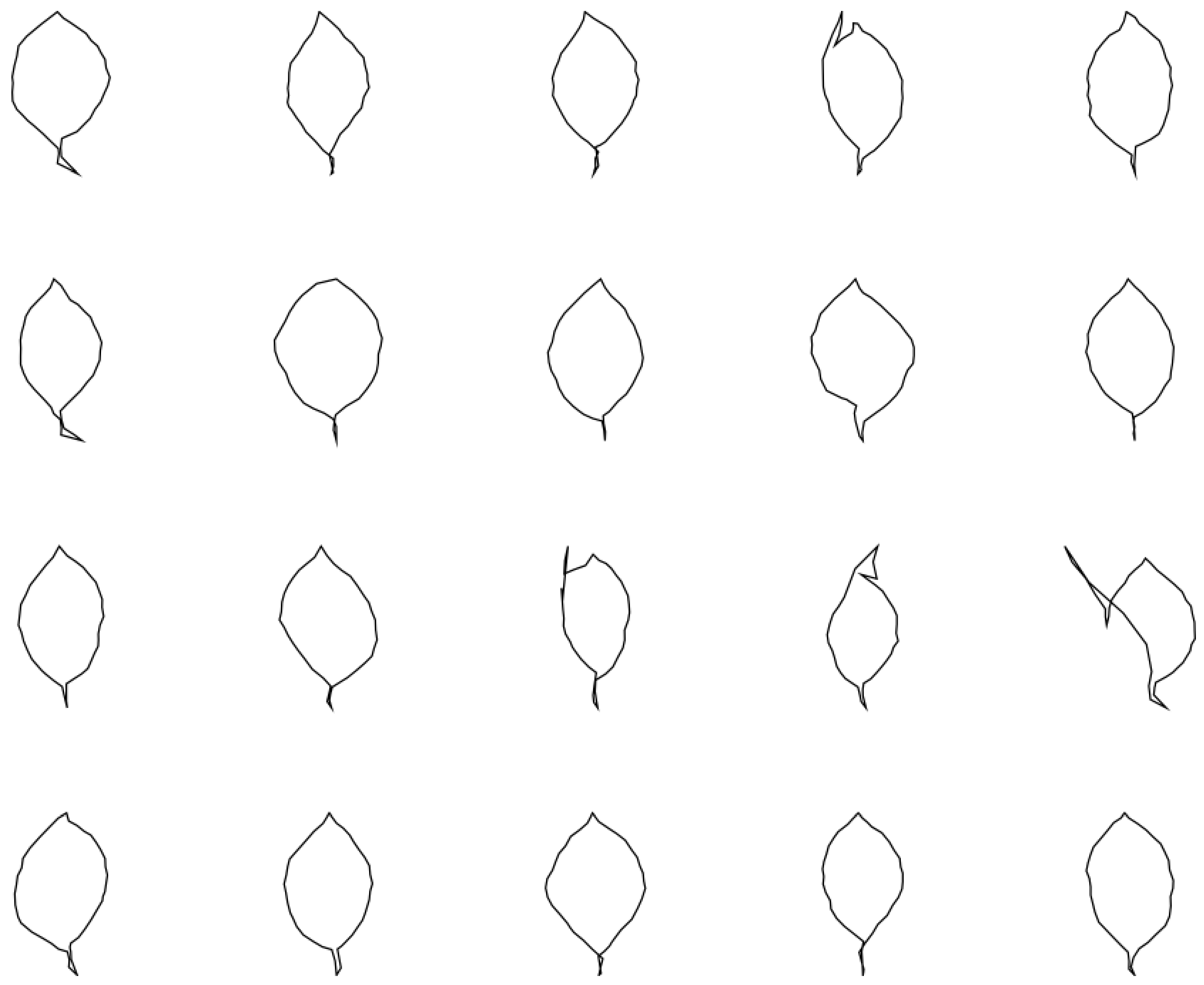


Figure A.30: Samples from learned model for class #15.

REFERENCES

Yali Amit and Alain Trouvé. Pop: Patchwork of parts models for object recognition. *International Journal of Computer Vision*, 75(2):267–282, November 2007. ISSN 0920-5691. doi: 10.1007/s11263-006-0033-9. URL `http://dx.doi.org/10.1007/s11263-006-0033-9`.

Moshe Bar. Visual objects in context. *Nature Reviews Neuroscience*, 5(8): 617–629, August 2004. ISSN 1471-003X. doi: 10.1038/nrn1476. URL `http://dx.doi.org/10.1038/nrn1476`.

R. Basri, L. Costa, D. Geiger, and D. Jacobs. Determining the similarity of deformable shapes. *Vision Research*, 38(15-16):2365–2385, August 1998. doi: 10.1016/S0042-6989(98)00043-1. URL `http://dx.doi.org/10.1016/S0042-6989(98)00043-1`.

E. J. Bernstein and Y. Amit. Part-based statistical models for object classification and detection. volume 2, pages 734–740 vol. 2, 2005. doi: 10.1109/CVPR.2005.270. URL `http://dx.doi.org/10.1109/CVPR.2005.270`.

Julian Besag. Statistical analysis of dirty pictures*. *Journal of Applied Statistics*, 20(5):63–87, 1993. doi: 10.1080/02664769300000059. URL `http://dx.doi.org/10.1080/02664769300000059`.

Elie Bienenstock, Stuart Geman, and Daniel Potter. Compositionality, mdl priors, and object recognition. In *Neural Information Processing Systems*, pages 838–844. MIT Press, 1997.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, 2003. ISSN 1533-7928. URL `http://portal.acm.org/citation.cfm?id=944919.944937`.

J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, November 1986. ISSN 0162-8828. doi: 10.1109/TPAMI.1986.4767851. URL `http://dx.doi.org/10.1109/TPAMI.1986.4767851`.

Eugene Charniak. *Statistical Language Learning (Language, Speech, and Communication)*. The MIT Press, September 1996. ISBN 0262531410.

C. Cook. Grammatical inference by hill climbing. *Information Sciences*, 10 (1):59–80, 1976. ISSN 00200255. doi: 10.1016/0020-0255(76)90061-X. URL `http://dx.doi.org/10.1016/0020-0255(76)90061-X`.

T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham. Active shape models - their training and application. *Comput. Vis. Image Underst.*, 61(1):38–59, Jan 1995. ISSN 1077-3142. doi: 10.1006/cviu.1995.1004.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977. ISSN 00359246. doi: 10.2307/2984875. URL `http://dx.doi.org/10.2307/2984875`.

I. L. Dryden and Kanti V. Mardia. *Statistical Shape Analysis*. Wiley, 1 edition, September 1998. ISBN 0471958166.

Fellbaum. *WordNet: An Electronic Lexical Database (Language, Speech, and Communication)*. The MIT Press, illustrated edition edition, May 1998. ISBN 026206197X.

P. Felzenszwalb and D. Huttenlocher. Pictorial structures for object recognition, 2003. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.12.6365`.

P. Felzenszwalb and J. Schwartz. Hierarchical matching of deformable shapes. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, 2007. doi: 10.1109/CVPR.2007.383018. URL `http://dx.doi.org/10.1109/CVPR.2007.383018`.

P. F. Felzenszwalb. Representation and detection of deformable shapes. *Pattern Analysis and*

Machine Intelligence, IEEE Transactions on, 27(2):208–220, February 2005. ISSN 0162-8828. doi: 10.1109/tpami.2005.35. URL `http://dx.doi.org/10.1109/tpami.2005.35`.

Pedro Felzenszwalb. *Representation and Detection of Shapes in Images.* PhD thesis, Massachusetts Institute of Technology, September 2003.

Pedro F. Felzenszwalb and David McAllester. The generalized a* architecture. *J. Artif. Int. Res.*, 29(1):153–190, 2007. ISSN 1076-9757. URL `http://portal.acm.org/citation.cfm?id=1622612`.

Pedro F. Felzenszwalb and David McAllester. Object detection grammars. Technical report, University of Chicago, February 2010. URL `http://www.cs.uchicago.edu/files/tr_authentic/TR-2010-02.pdf`.

K. S. Fu. A step towards unification of syntactic and statistical pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(3): 398–404, May 1986. ISSN 0162-8828. doi: 10.1109/TPAMI.1986.4767800. URL `http://dx.doi.org/10.1109/TPAMI.1986.4767800`.

Stuart Geman and Donald Geman. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-6(6):721–741, November 1984. ISSN 0162-8828. doi: 10.1109/TPAMI.1984.4767596. URL `http://dx.doi.org/10.1109/TPAMI.1984.4767596`.

Stuart Geman, Daniel F. Potter, and Zhiyi Chi. Composition systems. Technical report, Brown University. URL `http://www.cs.umd.edu/~djacobs/CMSC828/Composition%2520Systems.pdf`.

U. Grenander, Y. Chow, and D. M. Keenan. *Hands: A pattern Theoretic Study of Biological Shapes*, volume 2 of *Research Notes in Neural Computing.* Springer, New York, 1991.

Ulf Grenander and Michael Miller. *Pattern Theory: From Representation to Inference.* Oxford University Press, Inc., New York, NY, USA, 2007. ISBN 0199297061, 9780199297061. URL `http://portal.acm.org/citation.cfm?id=1512854`.

Feng Han and Song C. Zhu. Bottom-up/top-down image parsing with attribute grammar. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(1):59–73, 2009. ISSN 0162-8828. doi: 10.1109/TPAMI.2008.65. URL `http://dx.doi.org/10.1109/TPAMI.2008.65`.

D. Huttenlocher, D. Klanderman, and A. Rucklige. Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863, September 1993. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.2238`.

Ya Jin and S. Geman. Context and hierarchy in a probabilistic image model. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2145–2152, 2006. doi: 10.1109/CVPR.2006.86. URL `http://dx.doi.org/10.1109/CVPR.2006.86`.

Mark Johnson, Thomas L. Griffiths, and Sharon Goldwater. Bayesian inference for pcfgs via markov chain monte carlo. In *In Proceedings of the North American Conference on Computational Linguistics*, 2007. URL `http://acl.ldc.upenn.edu/N/N07/N07-1018.pdf`.

Benjamin Kimia, Ilana Frankel, and Ana-Maria Popescu. Euler spiral for shape completion. *International Journal of Computer Vision*, 54(1):159–182, August 2003. doi: 10.1023/A:1023713602895. URL `http://dx.doi.org/10.1023/A:1023713602895`.

Lillian Lee. Learning of context-free languages: A survey of the literature. Technical Report TR-12-96, Harvard University, 1996. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.40.7989`.

H. Ling and D. Jacobs. Shape classification using the inner-distance. *IEEE Trans-*

actions on Pattern Analysis and Machine Intelligence, 29:286–299, 2007. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.83.814.

Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1 edition, June 1999. ISBN 0262133601.

David Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. W. H. Freeman, March 1983. ISBN 0716715678.

David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, 2001. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.22.7314.

Craig G. Nevill-Manning and Ian H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm, 1997. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.3846.

E. Parzen. On the estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33:1065–1076, 1962. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.8666.

Daniel F. Potter. *Compositional pattern recognition*. PhD thesis, Providence, RI, USA, 1999. URL http://portal.acm.org/citation.cfm?id=929684.

Aristid Prusinkiewicz, Przemyslaw; Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990. ISBN 9780387972978.

C. Raphael. Coarse-to-Fine dynamic programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(12):1379–1390, 2001. ISSN 0162-8828. doi: 10.1109/34.977562. URL http://dx.doi.org/10.1109/34.977562.

Whitman Richards, Donald D. Hoffman, and Codon C. Richards. Codon constraints on closed 2d shapes. In *Computer Vision, Graphics, and Image Processing*, volume 31, pages 265–281, 1985. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.5639`.

Bryan C. Russell, Antonio Torralba, Kevin P. Murphy, and William T. Freeman. LabelMe: A database and web-based tool for image annotation. *International Journal of Computer Vision*, 77(1):157–173, May 2008. ISSN 0920-5691. doi: 10.1007/s11263-007-0090-8. URL `http://dx.doi.org/10.1007/s11263-007-0090-8`.

O Söderkvist. Computer vision classification of leaves from swedish trees. Master's thesis, Linkoping University, 2001.

Andreas Stolcke. *Bayesian learning of probabilistic language models*. PhD thesis, Berkeley, CA, USA, 1994. URL `http://portal.acm.org/citation.cfm?id=221997`.

Zhuowen Tu, Xiangrong Chen, Alan Yuille, and Song-Chun Zhu. Image parsing: Unifying segmentation, detection, and recognition. *International Journal of Computer Vision*, 63(2):113–140, July 2005. ISSN 0920-5691. doi: 10.1007/s11263-005-6642-x. URL `http://dx.doi.org/10.1007/s11263-005-6642-x`.

M. Wertheimer. *Laws of Organization in Perceptual Forms (partial translation)*, pages 71–88. Harcourt, Brace, 1938. URL `http://psychclassics.asu.edu/Wertheimer/Forms/forms.htm`.

Long Zhu, Yuanhao Chen, and Alan Yuille. Unsupervised learning of probabilistic grammar-markov models for object categories. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(1):114–128, 2009. ISSN 0162-8828. doi: 10.1109/TPAMI.2008.67. URL `http://dx.doi.org/10.1109/TPAMI.2008.67`.

Song C. Zhu and David Mumford. A stochastic grammar of images. *Found. Trends. Com-*

put. Graph. Vis., 2(4):259–362, 2006. ISSN 1572-2740. doi: 10.1561/0600000018. URL `http://dx.doi.org/10.1561/0600000018`.