# Chapter 2
# Multi-domain Hierarchical Free-Sketch Recognition Using Graphical Models

**Christine Alvarado**

## 2.1 Introduction

Consider the following physics problem:

> An 80-kg person is standing on the edge of a 3.6-m cliff. A 3-meter rope is attached to a point directly above his head, and on the end of the rope is a 40-kg medicine ball. The ball swings down and knocks the person off the cliff. Fortunately, there is a (padded) cart at the bottom. How far away from the cliff must the cart be placed in order to catch the person?

The above problem illustrates the central role of pictures and diagrams in understanding. It is almost impossible to read this problem without picturing the scenario in your head, and, for most people, the diagram is essential in solving the problem.

Because of the power of diagrams in thinking and design [8, 12, 56], people rely heavily on hand-sketched diagrams as a quick, lightweight way to put their ideas on paper and help them visualize solutions to their problems. Students, designers, scientists and engineers use sketches in a wide variety of domains, from physical (e.g., mechanical and electrical engineering designs) to conceptual (e.g., organizational charts and software diagrams).

Diagrams drawn on paper are just static pictures, but when drawn on a tablet computer, diagrams have the potential to be interpreted by the computer, and then made interactive. With the rise of pen-based technologies, the number of sketch-based computer tools is increasing. Sketch recognition-based computer systems have been developed for a variety of domains including (but not limited to) mechanical engineering [2, 21, 51], electrical engineering [16], user interface design [9, 34, 42], military course of action diagrams [11, 13], mathematical equations [33, 41], physics [39], musical notation [7, 14], software design [24, 36], note taking (Microsoft OneNote), and image editing [46]. In addition, a few multi-domain recognition toolkits have been proposed [3, 25, 35, 40].

C. Alvarado (✉)
Harvey Mudd College, 301 Platt Blvd., Claremont, CA, USA
e-mail: alvarado@cs.hmc.edu

J. Jorge, F. Samavati (eds.), *Sketch-based Interfaces and Modeling*,
DOI 10.1007/978-1-84882-812-4_2, 

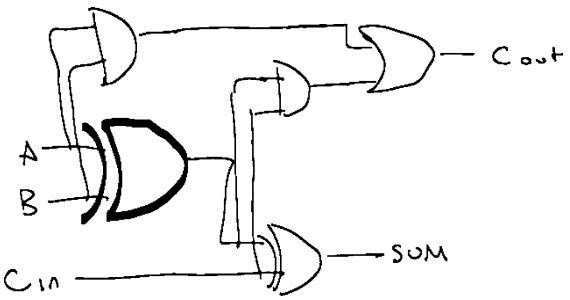



**Fig. 2.1** A diagram drawn by a student in a digital design class (stroke thickness altered for illustration)

The problem of two-dimensional sketch recognition is to parse the user's strokes to determine the best set of known patterns to describe the input. This process involves solving two interdependent subproblems: stroke segmentation and symbol recognition. *Stroke segmentation* (or just *segmentation*) is the process of determining which strokes should be grouped to form a single symbol. *Symbol recognition* is the process of determining what symbol a given set of strokes represents.

Despite the growing number of systems, this two-dimensional parsing problem remains a challenging problem for a real-time system. Sketched symbols rarely occur in their canonical form: both noise in the sketch and legal symbol variations make individual symbols difficult to recognize. Furthermore, segmentation and symbol recognition are inherently intertwined. In the sketch in Fig. 2.1, if the system could correctly group the three bold strokes in this sketch, it likely could identify those strokes as an XOR gate using a standard pattern matching technique. Unfortunately, simple spatial and temporal grouping approaches do not work: the three strokes that form the XOR gate are not all touching each other, but they are touching the input and output wires. If the computer somehow can find the correct grouping, it probably will be able to match the strokes to a shape in its library. However, naïvely trying all combinations of stroke groups is prohibitively time-consuming.

Researchers have employed different techniques to cope with these challenges. Some of the systems listed above perform only limited recognition by design. ScanScribe, for example, uses perceptual guidelines to support image and text editing but does not attempt to recognize the user's drawing [46]. Similarly, the sketch-based DENIM system supports the design of web pages but recognizes very little of the user's sketch [42]. Systems of this sort are powerful for their intended tasks, but they do not support a the creative sketch-based design process in more complex domains.

Other recognition systems place restrictions on the user's drawing style in order to make recognition easier. We list four common drawing style restrictions that address these challenges, ordered from most restrictive to least restrictive, and give examples of systems that use each:

1. Users must draw each symbol using a pre-specified pattern or gesture (e.g., Palm Graffiti®, ChemPad [54]).
2. Users must trigger recognition after each symbol (or pause notably between symbols) (e.g., HHreco [28], QuickSet [11]).
3. Users must draw each symbol using temporally contiguous strokes (e.g., ACSPARC [16]).

4. Some systems place few restrictions on the way users draw, but rely on user assistance or specific domain assumptions to aid recognition. To trigger recognition in MathPad$^2$, for example, the user must circle pieces of the sketch [39]. The approach presented by Kara and Stahovich performs robust recognition of feedback control system diagrams, but relies on the assumption that the diagram consists of a number of shapes linked by arrows, which is not the case in many other domains [31].

While these previous systems have proven useful for their respective tasks, we aim to create a general sketch recognition system that does not rely on the drawing style assumptions of any one domain. This chapter describes a general-purpose recognition engine that can be applied to a number of symbolic domains by inputting the shapes and commonly occurring combinations of shapes using a hierarchical shape description language, described below. Based on these descriptions, we use a constraint-based approach to recognition, evaluating potential higher-level interpretations for the user's strokes by evaluating their subcomponents and the constraints between them. To achieve recognition robustness and efficiency, we use a combined bottom–up and top–down recognition algorithm that generates the most likely (possibly incomplete) interpretations first (bottom–up) and then actively seeks out lower-level parts of those interpretations that are still missing (top–down).

This chapter presents a synthesis of work presented in [3] and [4], as well as recent work that builds on this prior work. We begin by exploring the challenges of recognizing real-world sketches. Next, we present our approach to recognition, including how we represent knowledge in our system, how we manage uncertainty, and our method of searching for possible interpretations of the user's sketch. Next we analyze our system's performance on real data in two domains. We conclude with a discussion of the major remaining challenge for multi-domain sketch recognition revealed by our evaluation: the problem of efficient and reliable sketch segmentation. We present an emerging technique that attempts to solve this problem.

## 2.2 The Challenges of Free-Sketch Recognition

Like handwriting and speech understanding, sketch understanding is easy for humans, but difficult for computers. We begin by exploring the inherent challenges of the task.

Figure 2.2 shows the beginning of a sketch of a family tree, with the strokes labeled in the order in which they were drawn. The symbols in this domain are given in Fig. 2.3. This sketch is representative of drawing patterns found in real-world data [5], but it has been redrawn to illustrate a number of challenges using a single example. The user started by drawing a mother and a father, then drew three sons. He linked the mother to the sons by first drawing the shafts of each arrow and then drawing the arrowheads. (In our family tree diagrams, each parent is linked to each child with an arrow.) He will likely continue the drawing by linking the father to the children with arrows and linking the two parents with a line.

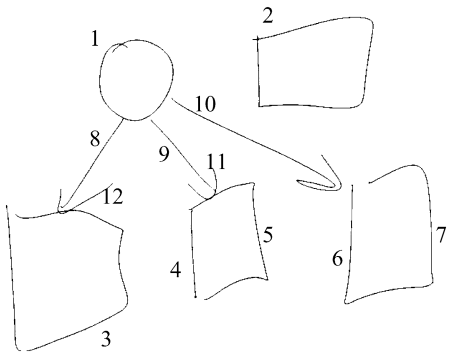



**Fig. 2.2** A partial sketch of a family tree

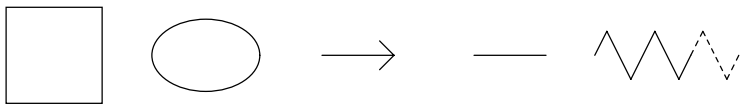



**Fig. 2.3** The symbols in the family tree domain

Although relatively simple, this drawing presents many challenges for sketch recognition. The first challenge illustrated in Fig. 2.2 is the incremental nature of the sketch process. Incremental sketch recognition allows the computer to seamlessly interpret a sketch as it is drawn and keeps the user from having to specify when the sketch is complete. To recognize a potentially incomplete sketch, a computer system must know when to recognize a piece of the sketch and when to wait for more information. For example, Stroke 1 can be recognized immediately as a female, but Stroke 6 cannot be recognized without Stroke 7.

The second challenge is that many of the shapes in Fig. 2.2 are visually messy. For example, the center arrowhead (Stroke 11) looks more like an arc than two lines. Next, the stroke used to draw the leftmost quadrilateral (Stroke 3) looks like it is composed of five lines—the top of the quadrilateral has a bend and could be reasonably divided into two lines by a stroke parser. Finally, the lines in the rightmost quadrilateral (Strokes 6 and 7) obviously do not touch in the top-left corner.

The third issue is segmentation: It is difficult to know which strokes are part of which shapes. The shapes in this drawing are not clearly spatially segmented, and naïvely trying different combinations of strokes is prohibitively time-consuming. There are also some inherent ambiguities in how to segment the strokes. For example, lines in our domain indicate marriage, but not every line is a marriage-link. The shaft of the leftmost arrow (Stroke 8) might also have been interpreted as a marriage-link between the female (Stroke 1) and the leftmost male (Stroke 3). In this case, the head of that arrow (Stroke 12) could have been interpreted as a part of the drawing that is not yet complete (e.g., the beginning of an arrow from the leftmost quadrilateral (Stroke 3) to the top quadrilateral (Stroke 2)).

Finally, how shapes are drawn can also present challenges to interpretation. The head of the right-most arrow (part of Stroke 10) is actually made of three lines, two of which are meant to overlap to form one side of the arrowhead. In order to recognize the arrow, the system must know how to collapse those two lines into one, even though they do not actually overlap. Another challenge arises because the same shape may not always be drawn in the same way. For example, the arrows on the left (Strokes 8 and 12, and Strokes 9 and 11) were drawn differently from the one on the right (Stroke 10) in that the user first drew the shaft with one stroke and then drew the head with another. This variation in drawing style presents a challenge for segmentation and recognition because a system cannot know how many strokes will be used to draw each object, nor the order in which the parts of a shape will appear.

Many of the difficulties described in the example above arise from the messy input and visual ambiguity in the sketch. It is the context surrounding the messy or ambiguous parts of the drawing that allows humans to interpret these parts correctly. We found that context also can be used to help our system recover from low-level interpretation errors and correctly identify ambiguous pieces of the sketch. Context has been used to aid recognition in speech recognition systems; it has been the subject of recent research in computer vision [52, 55] and has been used to some extent in previous sketch understanding systems [2, 16, 22, 42, 49, 50]. In the work presented here, we formalize the notion of context suggested by previous sketch recognition systems. This formalization improves recognition of freely-drawn sketches using a general engine that can be applied to a variety of domains.

## 2.3 Knowledge Representation

The goal of any recognition system is to match its input against an internal representation of a shape or set of shapes and identify the best match or matches (if any) for the given input. However, how each system represents the shape or shapes to be recognized (and consequently how each system matches the input to this internal representation) varies from system to system. For example, one system might represent each shape as a bit-mapped image template of the canonical form of that shape. Then, to perform recognition, that system would apply a series of legal transformations to the input data (e.g., rotation, scaling) to determine whether or not the pixels in the input can be made to line up with the pixels in the template. In contrast, a different system might represent each shape not as an image but as a collection of features extracted from the shapes. Examples of potential features include the ratio between the height and width of the bounding box of the shape, the total length of the strokes in the shape relative to the size of the bounding box, the number of corners in the shape, etc. Recognition in this system would then extract the same features from the input data and determine whether or not the features extracted from the input data are close enough to the features stored for each shape.

While many different representations can be used to perform recognition, the choice of internal shape representation affects the recognition task difficulty. In the

example above, recognition using the feature-based approach is more straightforward than the template-matching approach as it involves only a relatively small number of easy to calculate features rather than multiple transformations of the whole input. However, depending on the shapes in the domain, it may be extremely difficult to devise a set of features that reliably separates one shape from another.

Our system represents symbols to be recognized using a probabilistic, hierarchical description language. In choosing our representation, we considered several desired functionalities of our recognition system. First, the system should be extensible to new domains, requiring few training examples. Second, the system should be able to distinguish between legal and illegal shape transformations when performing recognition. Legal transformations include not only rotation, translation and scaling but also some non-rigid shape transformations. For example, the angle between a line in the head of an arrow and the shaft may range from about 10 degrees to about 80 degrees, but an angle greater than 90 degrees is not acceptable. Third, we would like to use this recognition system to compare various techniques for providing recognition feedback to the user, so the system should be able to recognize the sketch as the user draws to allow the system to potentially provide recognition feedback at any point in the drawing process. Finally, the system should be able to cope with the noise inherent in hand-drawn diagrams (e.g., lines that are not really straight, corners that do not actually meet, etc.).

This section describes our hierarchical description language and discusses how this choice of representation allowed us to construct a system that meets the requirements above. We begin by introducing the deterministic properties of the language, then discuss how uncertainty is incorporated into the descriptions. Finally, we discuss the advantages and disadvantages of this choice of representation.

### *2.3.1 Hierarchical Shape Descriptions*

Each shape in the domain to be recognized is described using a hierarchical description language, called LADDER, developed by Hammond and Davis [25]. We introduce the language through examples from the family tree and circuit domains.

We refer to any pattern recognizable in a given domain as a *shape*. *Compound shapes* are those composed of *subshapes*. Compound shapes must be non-recursive. Describing a compound shape involves specifying its subshapes and any necessary *constraints* between those subshapes. As an example, the description of an arrow is given in Fig. 2.4. The arrow has three subshapes—the line that is the shaft and the two lines that combine to make the head. The constraints specify the relative size, position and orientation necessary for these three lines to form an arrow shape (as opposed to just being three arbitrary lines). Once a shape has been defined, other shapes may use that shape in their descriptions. For example, the child-link symbol in the family tree domain (Fig. 2.5) and the current source symbol in the circuit domain (Fig. 2.6) both use an arrow as a subshape.

Shapes that cannot be broken down into subshapes are called *primitive shapes*. The set of primitive shapes includes free-form strokes, lines, arcs and ellipses.

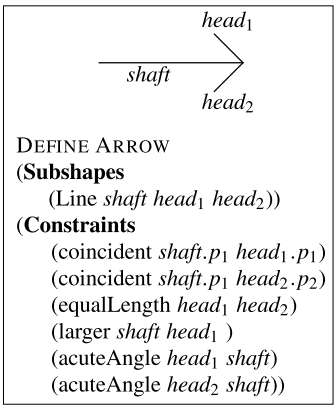



**Fig. 2.4** The description of the shape "arrow." Once defined, this shape can be used in descriptions of domain-specific shapes, as in Figs. 2.5 and 2.6

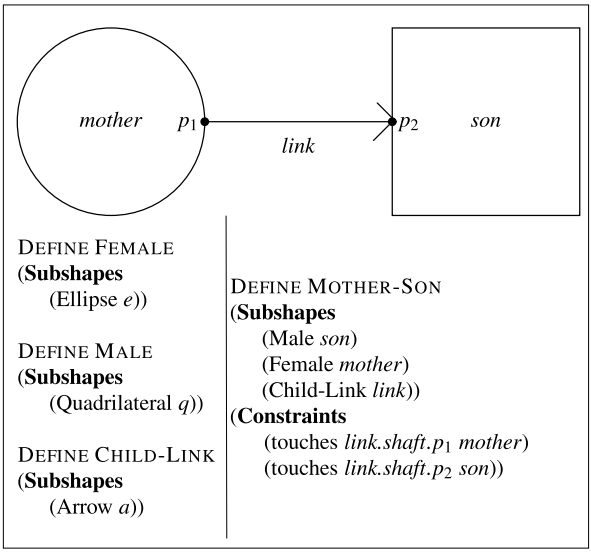



**Fig. 2.5** Descriptions of several domain shapes (Female, Male and Child-Link) and one domain pattern (Mother–Son) in the family tree domain

Although primitive shapes cannot be decomposed into subshapes, they may have named *subcomponents* that can be used when describing other shapes, e.g., the end-points of a line, $p_1$ and $p_2$, used in Fig. 2.4.

*Domain shapes* are shapes that have semantic meaning in a particular domain. Child-link and current-source are both domain shapes, but arrow and line are not because they are not specific to any one domain. *Domain patterns* are combinations of domain shapes that are likely to occur, for example the child-link pointing from a female to a male, indicating a relationship between mother and son in Fig. 2.5. Compound shape descriptions with no constraints (e.g., the child-link description) are used to rename a generic geometric shape (e.g., the arrow) as a domain shape so that domain-specific semantics may be associated with the shape.

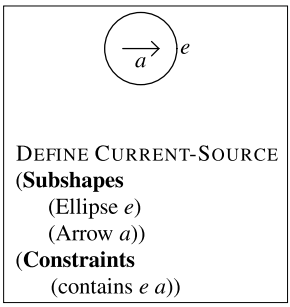



**Fig. 2.6** The description of a Current Source from the circuit domain

### *2.3.2 Handling Noise in the Drawing*

The system's goal in recognition is to choose the best set of domain shapes for a given set of strokes. While this task appears to be straightforward, Sect. 2.2 illustrated that ambiguity in the drawing can make recognition more difficult. Here, we describe the language constructs that help the system cope with the inevitable noise and ambiguities in the drawing. We discuss two different types of variation supported by our representation: signal-level noise and description-level variation.

#### 2.3.2.1 Signal-Level Noise: Objective vs. Subjective Measures

Shape descriptions specify the subshapes and constraints needed to form a higher-level shape; however, people rarely draw shapes perfectly or constraints that hold exactly. For example, although a user intends to draw two parallel lines, it is unlikely that these lines will be exactly parallel. We call this type of variation *signal-level noise*.

Because of signal-level noise, low-level shape and constraint interpretations must be based both on the data and on the context in which that shape or constraint appears. Consider whether or not the user intended for the two bold lines in each drawing in Fig. 2.7 to connect. In Figs. 2.7(a) and (b), the bold lines are identically spaced, but the context surrounding them indicates that in Fig. 2.7(b) the user intended for them to connect, while in Fig. 2.7(a) the user did not. On the other hand, the stroke information should not be ignored. The thin lines in Figs. 2.7(b) and (c) are identical, but the distance between the endpoints of the bold lines in these figures indicate that the these lines are intended to connect in Fig. 2.7(b) but not in Fig. 2.7(c).

For each low-level shape and constraint we identify an objectively measurable property that corresponds to that shape or constraint. For example, the property related to the constraint *coincident* is the distance between the two points in question normalized by the length of the lines containing the points in question. This objectively measurable property allows the system to separate the information provided by the stroke data from the information provided by the surrounding context to determine whether or not the constraint actually holds. Section 2.5 discusses precisely how these low-level measurements and the contextual data are combined.

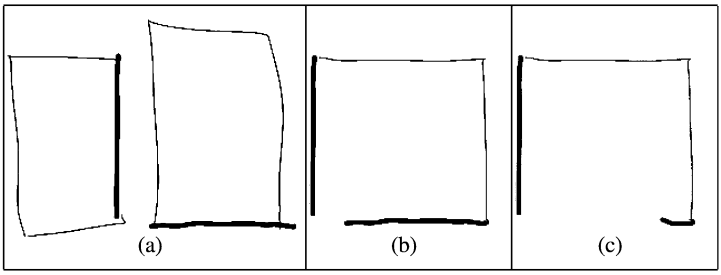
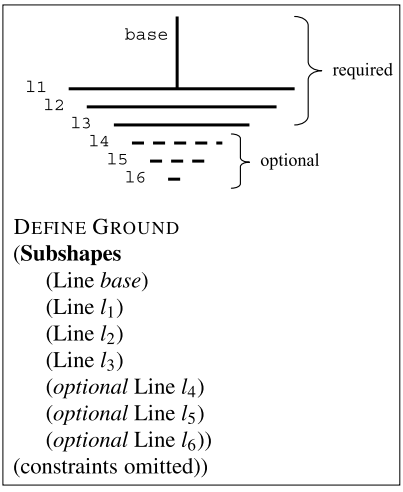



**Fig. 2.7** The importance of both data and context in determining whether or not the *bold lines* were intended to connect

**Fig. 2.8** The ground symbol from the Circuit Diagram domain




### 2.3.2.2 Description-Level Variation: Optional Components and Constraints

All of the shapes considered so far have been modeled using a fixed number of subshapes and a set of required constraints between those subshapes. These descriptions signify that, when a user draws these symbols, she should draw all of the subparts specified.

In contrast, some shapes have subcomponents that can be omitted legally when they are drawn. For example, consider the ground symbol described in Fig. 2.8. The user may draw up to six horizontal lines, but three of these lines optionally may be omitted. These three lines are flagged as *optional* in the shape description. We call this type of variation *description-level variation*.

Constraints may also be flagged as optional, indicating that they often hold, but are not required in the description of a symbol. For example, we could define the domain shape *wire* as having the single subshape *line* and the optional constraint that the line is horizontal or vertical. This constraint is not strictly required, as wires may be drawn diagonally, but they are often drawn either horizontally or vertically.

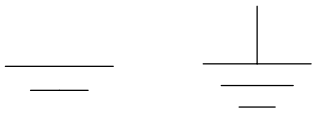


**Fig. 2.9** Battery (*left*) and ground (*right*) symbols from the Circuit Diagram domain. Note that the battery symbol is a subset of the ground symbol

Constraints pertaining to optional components are considered to be required if the optional component is present unless they are explicitly flagged as optional.

Understanding the difference between signal-level noise and description-level variation is central to understanding our representation of uncertainty. Signal-level noise is distinguished from description-level variation by considering the user's intent when she draws a symbol. For example, in a ground symbol the *base* line should be perpendicular to line $l_1$. In a given drawing, the angle between those lines may actually be far from 90 degrees due to signal-level noise (which might be caused by the user's sloppiness), but the lines are still intended to be perpendicular. On the other hand, the ground symbol may not contain line $l_6$, not because the user was being sloppy, but because the user did not intend to include it when drawing the symbol. We discuss how we model each variation in Sect. 2.5.

### *2.3.3 Strengths and Limitations*

We chose this symbolic, hierarchical representation based on the recognition system guidelines presented in the first part of this section. Here, we consider how this representation supports the creation of a multi-domain free-sketch recognition system. As every representation choice has trade-offs, we also consider the limitations of this approach and briefly discuss how these limitations can be addressed.

The first requirement was that our system must be extensible to new domains, requiring few training examples. To extend the system to a new domain, a user must simply describe the domain shapes and patterns for the new domain. Because the system can use the same hierarchical recognition process, it does not need to be trained with a large number of examples for each new shape. Furthermore, basic geometric shapes can be defined once and reused in a number of domains.

The second requirement was that our system must accept legal non-rigid transformations of sketched symbols without accepting illegal transformations. This requirement is handled by the fact that constraints can be defined to accept a wide range of relationships between shapes. For example, *acuteAngle* refers to any angle less than 90 degrees. Furthermore, only constraints explicitly stated in the shape definition are verified. Constraints that are not specified may vary without affecting the system's interpretation of the shape. For example, the definition of a quadrilateral would not constrain the relative lengths of the lines or the angles between those lines, leaving the system free to interpret any set of four correctly connected lines as a quadrilateral.

The third requirement was that the system be able to recognize a sketch as it is being drawn. To support this goal, our representation in terms of a shape's subcomponents allows the system to detect when it has seen only some of the subcomponents of a given shape. Using these partial interpretations, the system can decide which interpretations are likely complete and which might still be in the process of being drawn. This capability is particularly important when shape descriptions overlap, as in the battery and the ground symbol in the circuit domain (Fig. 2.9). When the user draws what could look like a battery or part of a ground symbol, the system can detect the partial ground interpretation and wait until the user has drawn more strokes to give its final interpretation instead of immediately interpreting the strokes as a battery.

The final requirement was that our system must deal with the noise in hand-drawn diagrams. Our representation allows us to handle signal-level noise by separating low-level objective measurements from judgements about whether or not constraints hold.

Although our representation satisfies the above requirements, it also imposes some restrictions. First, even with a well designed language, specifying shape descriptions may be difficult or time-consuming. Others have developed systems to learn shape descriptions from few examples. In work by Vesselova and Davis [57], as the user draws a shape, the learning system parses the user's strokes into low level components such as lines, arcs, and ellipses. The learner then calculates the existing constraints between these components and uses perceptual cues to deduce which constraints are most important the shape description. Once the system has learned a shape (e.g., a rectangle) it can then use that shape in its description of other shapes (e.g., a house). Hammond and Davis extended this work with an interactive system that helps users debug shape descriptions generating and displaying "near-miss" examples (i.e. shape patterns that vary only slightly from the current description) [27]. The output of both systems is a description of a domain shape in the visual language.

Second, even if we could build a system to learn shape descriptions, some shapes may be difficult or impossible to describe in terms of any simple low-level components or constraints. Those domains with free-form shapes, such as architecture, may have many shapes that cannot be easily described. Our representation is appropriate only for domains with highly structured symbols.

Finally, using this representation it is difficult to represent text or unrecognized strokes. This limitation must be addressed in the recognition system itself. The system should be capable of detecting text or unrecognized strokes and processing them using a different recognition technique or leaving them as unrecognized. Separating text from diagrams is a challenging problem that we do not address here, although recent approaches have proven quite successful at this task [6, 58].

## 2.4 Recognition Overview

As described above, a core challenge in two-dimensional sketch recognition is the problem of simultaneous segmentation and symbol recognition. Low-level inter-

pretations potentially can help guide the search for possible higher-level interpretations; for example, if the system detects two connected lines, it can first try to match a quadrilateral whose corner lines up with the connection between the lines. However, noise in the input makes it impossible for the system to recognize low-level shapes with certainty or to be sure whether or not constraints hold. Low-level misinterpretations cause higher-level interpretations to fail as well. Trying all possible interpretations of the user's strokes guarantees that an interpretation will not be missed, but it is infeasible due to the exponential number of possible interpretations.

To solve this problem we use a combined bottom–up and top–down recognition algorithm that generates the most likely interpretations first (bottom–up) and then actively seeks out parts of those interpretations that are still missing (top–down). Our approach uses a novel application of dynamically constructed Bayesian networks to evaluate partial interpretation hypotheses and then expands the hypothesis space by exploring the most likely interpretations first. The system does not have to try all combinations of all interpretations, but can focus on those interpretations that contain at least a subset of easily-recognizable subshapes and can recover any low-level subshapes that may have been mis-recognized.

We use a two-stage generate-and-test method to explore possible interpretations for the user's strokes. In the first stage, the system generates a number of *hypotheses*, or possible interpretations for the user's strokes, based on the shape descriptions described in Sect. 2.3. We refer to each shape description as a *template* with one *slot* for each subpart. A *shape hypothesis* is a template with an associated mapping between slots and strokes. Similarly, a *constraint hypothesis* is a proposed constraint on one or more of the user's strokes. A *partial hypothesis* is a hypothesis in which one or more slots are not bound to strokes. Our method of exploring the space of possible interpretations depends on our ability to assess both complete and partial hypotheses for the user's strokes. Section 2.5 describes our hypothesis evaluation technique; Sect. 2.6 describes how these hypotheses are generated.

## 2.5 Hypothesis Evaluation

We evaluate our shape hypotheses using dynamically constructed Bayesian networks specifically targeted to the task of constraint-based recognition. Our framework is closely related to previously proposed frameworks ([32, 37, 44]) but it was designed to handle the specific problems presented above that arise in the recognition task. Our method offers two advantages over previous constraint-based recognition approaches (e.g., [15, 20, 26]). First, missing data can be treated as unobserved nodes in the network when the system assesses likely hypotheses for the strokes that have been observed thus far. This allows our system to evaluate partial hypotheses (e.g., an arrow with no shaft) in order to interpret drawings as they develop, and to allow the strength of partial hypotheses to guide the interpretation of new strokes as they are processed. Second, the system's belief in a given hypothesis can be influenced both by the stroke data (through the node's children) and the context in which those shapes appear (through the node's parents), allowing the system to cope with noise in the drawing.

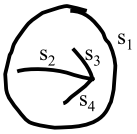
### 2.5.1 Dynamically Constructed Graphical Models

Time-based graphical models, including Hidden Markov Models (HMMs) and Dynamic Bayesian Networks (DBNs), have been applied successfully to time-series data in tasks such as speech understanding. To the extent that stroke order is predictable, HMMs and DBNs may be applied to sketch understanding (see [47] for one approach). Ultimately, however, sketch understanding is different because we must model shapes based on two-dimensional constraints (e.g., intersects, touches) rather than on temporal constraints (i.e., follows), and because our models cannot simply unroll in time as data arrive (we cannot necessarily predict the order in which the user will draw the strokes, and things drawn previously can be changed). Therefore, our network represents spatial relationships rather than temporal relationships.

It is not difficult to use Bayesian networks to model spatial relationships. The difficult part of using Bayesian networks for sketch understanding is that they are traditionally used to model static domains in which the variables and relationships between those variables are known in advance. Static networks are not suitable for the task of sketch recognition because we cannot predict a priori the number of strokes or symbols the user will draw in a given sketch. In fact, there are many tasks in which the possible number of objects and relationships may not be modeled a priori. For example, when reasoning about military activity, the number of military units and their locations cannot be known in advance. For such tasks, models to reason about specific problem instances (e.g., a particular sketch or a particular military confrontation) must be dynamically constructed in response to a given input. This problem is known as the task of *knowledge-based model construction* (KBMC).

A number of researchers have proposed models for the dynamic creation of Bayesian networks for KBMC. Early approaches focused on generating Bayesian networks from probabilistic knowledge bases [18, 19, 23, 45]. A recently proposed representation, called Network Fragments, represents generic template knowledge directly as Bayesian network fragments that can be instantiated and linked together at run-time [37]. Finally, Koller et al. have developed a number of a number of object-oriented frameworks including Object-Oriented Bayesian Networks (OOBNs) [32, 44] and Probabilistic Relational Models (PRMs) [17]. These models represent knowledge in terms of relationships among objects and can be instantiated dynamically in response to the number of objects in a particular situation.

Although the above frameworks are powerful, they are not directly suitable for sketch recognition because they are too general in some respects and too specialized in others. First, with this type of general model, it is a challenge simply to decide how to frame our recognition task in terms of objects, network fragments, or logical statements. Second, because these models are general, they do not make any assumptions about how the network will be instantiated. Because of the size of the networks potentially generated for our task, it is sometimes desirable to generate only part of a complete network, or to prune nodes from the network. In reasoning about nodes that are in the network, we must account for the fact that the network may not be fully generated or relevant information may have been pruned from the network. Finally, these models are too specific in that they have been optimized for




**CS hypothesis $CS_1$**
**Subshapes:**
$e$ → ellipse hypothesis $E_1$
$a$ → arrow hypothesis $A_1$
**Constraints:**
$C_1$ = (contains `ellipse-fit`($s_1$) `shape-fit`($s_2$, $s_3$, s4))

**Primitive hypotheses**
$E_1$ = `ellipse-fit`($s_1$)
$L_1$ = `line-fit`($s_2$)
$L_2$ = `line-fit`($s_3$)
$L_3$ = `line-fit`($s_4$)

**Arrow hypothesis $A_1$**
**Subshapes:**
*shaft* → line hypothesis $L_1$
$head_1$ → line hypothesis $L_2$
$head_2$ → line hypothesis $L_3$
**Constraints:**
$C_2$ = (coincident `line-fit`($s_2$).p1 `line-fit`($s_3$).p1)
$C_3$ = (coincident `line-fit`($s_2$).p1 `line-fit`($s_4$).p1)
$C_4$ = (equalLength `line-fit`($s_3$) `line-fit`(s4))
$C_5$ = (shorter `line-fit`($s_3$) `line-fit`($s_2$))
$C_6$ = (acuteAngle `line-fit`($s_2$) `line-fit`($s_3$))
$C_7$ = (acuteAngle `line-fit`($s_2$) `line-fit`($s_4$))

**Fig. 2.10** A single current source hypothesis ($CS_1$) and associated lower-level hypotheses. Shape descriptions for the arrow and current source (with labeled subshapes) are given in Figs. 2.6 and 2.4

responding to specific queries, for example, "What is the probability that a particular battery in our military force has been hit?" In contrast, our model must provide probabilities for a full set of possible interpretations of the user's strokes.

### 2.5.2 Shape Fragments: Evaluating a Single Hypothesis

Briefly, Bayesian networks consist of two parts: a Directed Acyclic Graph that encodes *which* factors influence one another, and a set of Conditional Probability Distributions which specify *how* these factors influence one another.[1] Each node in the graph represents something to be measured, and a link between two nodes indicates that the value of one node is directly dependent on the value of the other. Each node contains a conditional probability function (CPF), represented as a conditional probability table (CPT) for discrete variables, specifying how it is influenced by its parents.

To introduce our Bayesian network model, we begin by considering how to evaluate the strength of a single current source (CS) hypothesis for the stokes in Fig. 2.10. The description of a current source symbol is given in Fig. 2.6. Based on the hierarchical nature of the shape descriptions, we use a hierarchical method of hypothesis evaluation. Determining the strength of a particular current source hypothesis is a

[1] We provide enough background on Bayesian networks to give the reader a high-level understanding of our model. To understand the details, those unfamiliar with Bayesian networks are referred to [10] for an intuitive introduction and [30] for more details.

matter of determining the strengths of its corresponding lower-level shape and constraint hypotheses. A particular current source hypothesis, $CS_1$, specifies a mapping between the subparts in the current source description and the user's strokes via lower-level hypotheses for the user's strokes (Fig. 2.10). $E_1$ is an ellipse hypothesis for $s_1$, $A_1$ is an arrow hypothesis involving strokes $s_2$, $s_3$ and $s_4$ (through its line hypotheses), and $C_1$ is a constraint hypothesis that an ellipse fit for stroke $s_1$ contains strokes $s_2$, $s_3$, and $s_4$. $A_1$ is further broken down into three line hypotheses ($L_1$, $L_2$ and $L_3$) and six constraint hypotheses ($C_2, \ldots, C_7$) according to the description of the arrow (Fig. 2.4). Thus, determining the strength of hypothesis $CS_1$ can be transformed into the problem of determining the strength of a number of lower-level shape and constraint hypotheses.

#### 2.5.2.1 Network Structure

The Bayesian network for this recognition task is shown in Fig. 2.11. There is one node in the network for each hypothesis described above, and each of these nodes represents a Boolean random variable that reflects whether or not the corresponding hypothesis is correct. The nodes labeled $O_1, \ldots, O_{11}$ represent measurements of the stroke data that correspond to the constraint or shape to which they are linked. The variables corresponding to these nodes have positive real numbered values. For example, the variable $O_2$ is a measurement of the squared error between the stroke $s_1$ and the best fit ellipse to that stroke. The value of $O_2$ is a real number between 0 and the maximum possible error between any stroke and an ellipse fit to that stroke. The boxes labeled $s_1, \ldots, s_4$ are not part of the Bayesian network but serve to indicate the stroke or strokes from which each measurement, $O_i$, is taken (e.g., $O_2$ is measured from $s_1$). $P(CS_1 = t \mid ev)$ (or simply $P(CS_1 \mid ev)$)[2], where $ev$ is the evidence observed from the user's strokes, represents the probability that the hypothesis $CS_1$ is correct.

There are three important reasons why the links are directed from higher-level shapes to lower-level shapes instead of in the opposite direction. First, whether or not a higher-level hypothesis is true directly influences whether or not a lower-level hypothesis is true. For example, if the arrow hypothesis $A_1$ is true, then it is extremely likely that all three line hypotheses, $L_1, L_2, L_3$, are also true. Second, this representation allows us to model lower-level hypotheses as conditionally independent given their parents, which reduces the complexity of the data needed to construct the network. Finally, continuous valued variables are difficult to incorporate into a Bayesian network if they have discrete valued children. Our representation ensures that the measurement nodes, which have continuous values, will be leaf nodes. These nodes can be pruned when they do not have evidence, thus simplifying the inference process.

Each shape description constrains its subshapes only relative to one another. For example, an arrow may be made from *any* three lines that satisfy the necessary constraints. Based on this observation, our representation models a symbol's subshapes

[2]Throughout this section, $t$ means true, and $f$ means false.

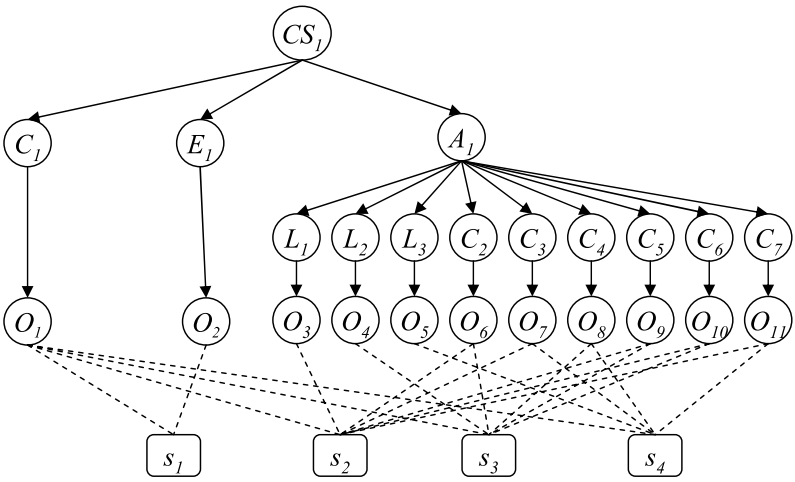



**Fig. 2.11** A Bayesian network to verify a single current source hypothesis. Labels come from Fig. 2.10

separately from the necessary constraints between those subshapes. For example, node $L_1$ represents the hypothesis that stroke $s_2$ is a line. Its value will be true if the user intended for $s_2$ to be any line, regardless of its position, size or orientation. Similarly, $C_2$ represents the hypothesis that the line fit to $s_2$ and the line fit to $s_3$ are coincident.

The conditional independence between subshapes and constraints might seem a bit strange at first. For example, whether or not two lines are of the same length seems to depend on the fact that they are lines. However, observation nodes for constraints are calculated in such a way that their value is not dependent on the true interpretation for a stroke. For example, when calculating whether or not two lines are parallel, which involves calculating the different in angle between the two lines, we first fit lines to the strokes (regardless of whether or not they actually look like lines), then measure the relative orientation of those lines. How well these lines fit the strokes is not considered in this calculation.

The fact that the shape nodes are not directly connected to the constraint nodes has an important implication for using this model to perform recognition: There is no guarantee in this Bayesian network that the constraints will be measured from the correct subshapes because the model allows subshapes and constraints to be detected independently. For example, $C_3$ in Fig. 2.11 indicates that $L_2$ and $L_3$ (the two lines in the head of an arrow) must be the same length, not simply that any two lines must have the same length. To satisfy this requirement, The system must ensure that $O_6$ is measured from the same strokes that $O_3$ and $O_4$ were measured from. We use a separate mechanism to ensure that only legal bindings are created between strokes and observation nodes.

The way we model shape and constraint information has two important advantages for recognition. First, this Bayesian network model can be applied to recognize a shape in any size, position and orientation. $CS_1$ represents the hypothesis that

$s_1, \ldots, s_4$ form a current source symbol, but the exact position, orientation and size of that symbol is determined directly from the stroke data. To consider a new hypothesis for the user's strokes, the system simply creates a copy of the necessary Bayesian network structure whose nodes represent the new hypotheses and whose measurement nodes are linked to a different set of the user's strokes. Second, the system can allow competing higher-level hypotheses for a lower-level shape hypothesis to influence one another by creating a network in which two or more hypotheses point to the same lower-level shape node. For example, the system may consider an arrow hypothesis and a quadrilateral hypothesis involving the same line hypothesis for one of the user's strokes. Because the line hypothesis does not include any higher-level shape-specific constraint information, both an arrow-hypothesis node and a quadrilateral hypothesis node can point to the single line hypothesis node. These two hypotheses then become alternate, competing explanations for the line hypothesis. We further discuss how hypotheses are combined below.

Our model is generative, in that we can use the Bayesian network for generation of values for the nodes in the network based on the probabilities in the model. However, our model is fundamentally different from the standard generative approach used in computer vision in which the system generates candidate shapes (for example, a rightward facing arrow) and then compares these shapes to the data in the image. The difference is that the lowest level of our network represents measurements of the strokes, not actual stroke data. So, although our model can be used to generate values of stroke data measurements, it cannot be used to generate shapes which can be directly compared to the user's strokes. However, because the system can always take measurements from existing stroke data, our model is well suited for hypothesis evaluation.

#### 2.5.2.2 Conditional Probability Distributions

Next, we consider the intuition behind the CPTs for a node given its parents for the hypotheses in Fig. 2.10. We begin by considering the distribution $P(E_1 \mid CS_1)$. Intuitively, we set $P(E_1 = t|CS_1 = t) = 1$ (and conversely, $P(E_1 = f \mid CS_1 = t) = 0$), meaning that if the user intended to draw $CS_1$, she certainly intended to draw $E_1$. This reasoning follows from the fact that the ellipse is a required component of the CS symbol (and that the user knows how to draw CS symbols). $P(E_1 \mid CS_1 = f)$, on the other hand, is a little less obvious. Intuitively, it represents the probability that the user intended to draw $E_1$ even though she did not intend to draw $CS_1$. This probability will depend on the frequency of ellipses in other symbols in the domain (i.e., higher if ellipses are common).

Because $CS_1$ has no parents, it must be assigned a prior probability. This probability is simply how likely it is that the user will draw $CS_1$. This probability will be high if there are few other shapes in the domain or if CS symbols are particularly prominent, and low if there are many symbols or if the CS symbol is rare. Exactly how these prior probabilities are determined is beyond the scope of this chapter but is discussed further in [1].

The bottom layer of the network accounts for signal-level noise by modeling the differences between the user's intentions and the strokes that she draws. For example, even if the user intends to draw $L_1$, her stroke likely will not match $L_1$ exactly, so the model must account for this variation. Consider $P(O_2 \mid E_1 = t)$. If the user always drew perfect ellipses, this distribution would be 1 when $O_2 = 0$, and 0 otherwise. However, most people do not draw perfect ellipses (due to inaccurate pen and muscle movements), and this distribution allows for this error. It should be high when $O_2$ is close to zero, and fall off as $O_2$ gets larger. The wider the distribution, the more error the system will tolerate, but the less information a perfect ellipse will provide.

The other distribution needed is $P(O_2 \mid E_1 = f)$ which is the probability distribution over ellipse error given that the user did not intend to draw an ellipse. This distribution should be close to uniform, with a dip around 0, indicating that if the user specifically does not intend to draw an ellipse, she might draw any other shape, but probably will not draw anything that resembles an ellipse. Discussion of how we determined the conditional probability distributions between primitive shapes and constraints and their corresponding measurement nodes can be found in [1].

#### 2.5.2.3 Observing Evidence from Stroke Data

Finally, we discuss how information from the user's strokes is incorporated into the network to influence the system's belief in $CS_1$. If we assume that the user is done drawing, the values of $O_1, \ldots, O_{11}$ are fully observable by taking measurements of the strokes. The system can then use those values to infer $P(CS_1 | O_1, \ldots, O_{11})$. If we do not assume the user is done drawing, we may still evaluate $P(CS_1 | ev)$ where the set $ev$ contains all $O_i$ corresponding to strokes the user has drawn so far.

We may model the current source (partial) hypothesis $CS_1$ even before the drawing is complete. An observation node that does not have an observed value intuitively corresponds to a stroke that the user has not yet drawn. Because observation nodes are always leaf nodes, the missing data have neither a positive nor a negative effect on the system's belief in a given interpretation. $CS_1$ may be strongly believed even if $O_1$ is missing. As described in Sect. 2.6, the system uses the strength of incomplete interpretations to help guide the search for missed low-level interpretations.

### *2.5.3 Recognizing a Complete Sketch*

The Bayesian network introduced above can be used to detect a single instance of a CS symbol in any size, position or orientation. However, a typical sketch contains several different symbols as well as several instances of the same symbol.

To detect other shapes in our domain, we may create a Bayesian network similar to the network above for each shape. We call each of these Bayesian networks

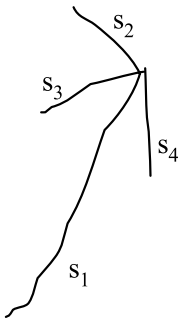



**Fig. 2.12** Four strokes, three of which form an arrow. The system might try both $s_2$ and $s_3$ as a line in the head of the arrow

a *shape fragment* because these fragments can be combined to create a complete Bayesian network for evaluating the whole sketch.

Above, we assumed that we were given a mapping between the user's strokes and the observation nodes in our network. In fact, the system must often evaluate a number of potential mappings between strokes and observation nodes. For example, if the user draws the four strokes in Fig. 2.12, the system might try mapping both $s_2$ and $s_3$ to $L_2$. As described above, each interpretation for a specific mapping between strokes and observation nodes is called a hypothesis, and each hypothesis corresponds to a single node in the Bayesian network. In this section we discuss how multiple hypotheses are combined to evaluate a complete sketch.

Given a set of hypotheses for the user's strokes, the system instantiates the corresponding shape fragments and links them together to form a complete Bayesian network, which we call the *interpretation network*. To illustrate this process, we consider a piece of a network generated in response to Strokes 6 and 7 in the example given in Fig. 2.2, which is reproduced in Fig. 2.13. Figure 2.14 shows the part of the Bayesian network representing the possible interpretations that the system generated for these strokes. Each node represents a hypothesized interpretation for some piece of the sketch. For example, $Q_1$ represents the system's hypothesis that the user intended to draw a quadrilateral with strokes 6 and 7. A higher-level hypothesis is compatible with the lower-level hypotheses it points to. For example, if $M_1$ (the hypothesis that the user intended to draw a male with strokes 6 and 7) is correct, $Q_1$ (the hypothesis that the user intended to draw a quadrilateral with strokes 6 and 7) and $L_1, \ldots, L_4$ (the hypotheses that the user intended to draw four lines with strokes 6 and 7) will also be correct. Two hypotheses that both point to the same lower-level hypothesis represent competing interpretations for the lower-level shape and are incompatible. For example, $A_1$, $Q_1$ are two possible higher-level interpretations for line $L_1$, only one of which may be true.

Each observation node is linked to a corresponding stroke or set of strokes. In a partial hypothesis, not all measurement nodes will be linked to stroke data. For example, $A_1$ is a partial hypothesis—it represents the hypothesis that $L_1$ and $L_2$ (and, hence, Stroke 6) are part of an arrow whose other line has not yet been drawn. Line nodes representing lines that have not been drawn ($L_5$ and $L_6$) are not linked to observation nodes because there is no stroke from which to measure these observations. We refer to these nodes (and their corresponding hypotheses) as *virtual*.

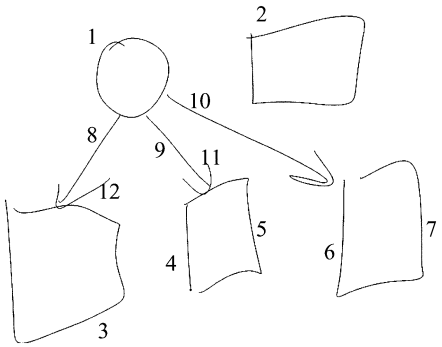



**Fig. 2.13** The partial sketch of a family tree from Sect. 2.1

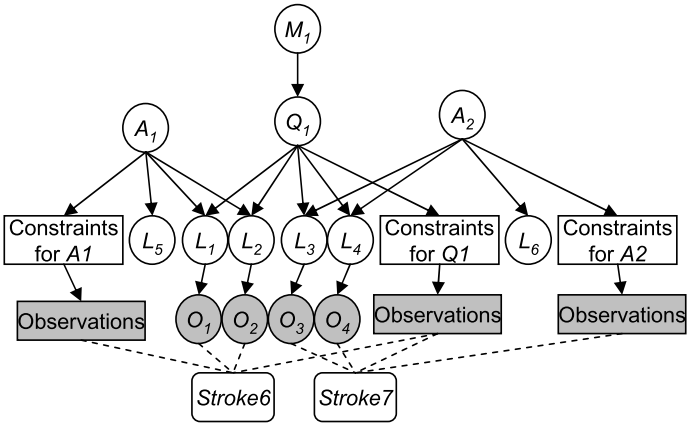



**Fig. 2.14** A portion of the interpretation network generated while recognizing the sketch in Fig. 2.13

The probability of each interpretation is influenced both by stroke data (through its children) and by the context in which it appears (through its parents), allowing the system to handle noise in the drawing. For example, there is a gap between the lines in the top-left corner in $Q_1$ (see Fig. 2.13); stroke data only weakly support the corresponding constraint hypothesis (not shown individually). However, the lines that form $Q_1$ are fairly straight, raising probabilities of $L_1, \ldots, L_4$, which in turn raise the probability of $Q_1$. $Q_1$ provides a context in which to evaluate the coincident constraint, and because $Q_1$ is well supported by $L_1, \ldots, L_4$ (and by the other constraint nodes), it raises the probability of the coincident constraint corresponding to $Q_1$'s top-left corner.

The fact that partial interpretations have probabilities allows the system to assess the likelihood of incomplete interpretations based on the evidence it has seen so far. In fact, even virtual nodes have probabilities, corresponding to the probability that the user (eventually) intends to draw these shapes but either has not yet drawn this part of the diagram or the correct low-level hypotheses have not yet been proposed because of low-level recognition errors. As we describe below, a partial interpre-

tation with a high probability cues the system to examine the sketch for possible missed low-level interpretations.

#### 2.5.3.1 Linking Shape Fragments

When Bayesian network fragments are linked during recognition, each node $H_n$ may have several parents, $S_1 \ldots S_m$, where each parent represents a possible higher-level interpretation for $H_n$. We use a noisy-OR function to combine the influences of all the parents of $H_n$ to produce the complete CPT for $P(H_n \mid S_1, \ldots, S_m)$. The noisy-OR function models the assumption that each parent can independently cause the child to be observed. For example, a single stroke might be part of a quadrilateral or an arrow, but both interpretations would favor that interpretation of the stroke as a line.

The intuition behind Noisy-OR is that each parent that is true will cause the child to also be true unless something prevents it from doing so. The probability that something will prevent a parent $S_i = t$ from causing $H_n = t$ to be true is $q_i = P(H_n = f \mid S_i = t)$. Noisy-OR assumes that all the $q_i$'s are independent, resulting in the following:

$$P(H_n = t \mid S_1, S_2, \ldots, S_m) = 1 - \prod_i q_i$$

for each $S_i = t$. We set $q_j = P(H_n = f \mid S_j = t) = 0$ for all parents $S_j$ in which $H_n$ is a required subshape or constraint, and we set $q_k = P(H_n = f \mid S_k = t) = 0.5$ for all parents $S_k$ in which $H_n$ is an optional subshape or constraint. A consequence of these values is that $S_j = t \Rightarrow P(H_n \mid S_1, \ldots, S_m) = 1$ for any $S_j$ in which $H_n$ is required, which is exactly what we intended.

Noisy-OR requires that $P(H_n \mid S_1 = S_2, \ldots, S_m = f]) = 0$. The result of this requirement is that any shape or constraint has zero probability of appearing if it is not part of a higher-level shape or pattern. This behavior may be what is desired; however, if it is not, we may create an additional parent, $S_a$, to model the probability that the user intends to draw $H_n$ alone, not as part of a shape or pattern.

#### 2.5.3.2 Missing Nodes

Throughout this process, we have assumed that all of the hypothesized interpretations will exist as a node in the Bayesian network. However, for reasons discussed in Sect. 2.6, there are two reasons a hypothesis might be missing from the network. First, the system does not always initially generate all higher-level interpretations for a shape. Second, the system prunes unlikely hypotheses from the network to control the network's size.

We would like hypotheses that have not yet been generated or that have been pruned nevertheless to influence the strength of the hypotheses in the network. For

example, if there are two potential interpretations for a stroke—a line and an arc—and the system prunes the line interpretation because it is too unlikely, the probability of the arc should go up. On the other hand, if the system has only generated an arc interpretation, and has not yet considered a line interpretation, the probability of the arc should remain modest because the stroke might still be a line.

We model nodes not present in the network through an additional parent, $S_{np}$, for each node $H_n$ in the graph. We define $q_{np} = P(H_n = f \mid S_{np} = t)$ and set $S_{np} = t$. The value of $q_{np}$ takes into account which nodes have been eliminated from the graph and which have not (yet) been instantiated, and it is calculated as follows. Let $T_1, \ldots, T_p$ be the set of shapes that have an element of type $H_n$ as a child but do not exist as parents of $H_n$ in the network. We refer to $T_1, \ldots, T_p$ as *potential parents* of $H_n$. For example, for node $L_1$ in Fig. 2.14, this set would contain the single element $ML$ because the marriage-link is the only shape in the family tree domain that has a line as a subshape but is not already a parent of $L_1$ in the graph. Let $T_1, \ldots, T_i$ be the subset of potential parents that have never appeared as a parent for $H_n$, and let $T_{i+1}, \ldots, T_p$ be the subset with elements that were once parents for $H_n$ but have been pruned from the network. Then, $q_{np} = \prod_{j=1}^{i} 1 - P(T_j)$. We call $P(T_j)$ the *simple marginal probability* of $T_j$. It is calculated by calculating the marginal probability based only on the priors of the parents and ancestors of $T_j$ in a network containing exactly one instance of each parent of $T_j$.

The effect of $q_{np}$ is to allow only those shapes that have not yet been instantiated as parents of $H_n$ to contribute to the probability that $H_n = t$. If the simple marginal probabilities of the missing parents are high, $q_{np}$ will be low, and thus will help raise $P(H_n \mid S_1, \ldots, S_m, S_{np})$. If all the potential parents of $H_n$ have previously been pruned, $q_{np}$ will be 1 and thus have no effect on $P(H_n \mid S_1, \ldots, S_m, S_{np})$.

### 2.5.4 Implementation and Bayesian Inference

Our system updates the structure of the Bayesian network in response to each stroke the user draws. To perform this dynamic Bayesian network construction, we use an off-the-shelf, open source Bayesian network package for Java called BNJ [61]. Our system manages the hypotheses for the user's strokes as they are generated and pruned. When these hypotheses need to be evaluated (e.g., before they are pruned), our system creates a Bayesian network in BNJ by creating the necessary nodes and links as BNJ Java objects. Our system can then use a number of methods that are built into BNJ for reasoning about the probability of each node.

Generating and modifying the BNJ networks can be time-consuming due to the exponential size of the conditional probability tables (CPTs) between the nodes. We use two techniques to improve the system's performance. First, BNJ networks are only generated when the system needs to evaluate the likelihood of a given hypothesis. This on-demand construction is more efficient than continuously updating the BNJ network because batch construction of the CPTs is often more efficient than incremental construction of these tables. Second, the system modifies only the portion of the BNJ network that has changed between strokes instead of creating it from

scratch every time. The process of keeping track of the added and removed hypotheses adds a slight bookkeeping overhead, but this overhead is far less than the work required to regenerate the entire network after each stroke.

To determine the likelihood of each hypothesis, our system uses the BNJ network to find the marginal posterior probability for each node in the network. We experimented with several inference methods including the junction tree algorithm [29, 38], Gibbs sampling, and loopy belief propagation (loopy BP) [43, 59]. Both the junction tree algorithm and loopy BP produced meaningful marginal posterior probabilities, but after some experimentation we were unable to obtain useful results using Gibbs Sampling. We discovered that although the junction tree algorithm gave meaningful results, the networks produced by our system were often too complex for the algorithm to process in a reasonable amount of time. We found that when the junction tree algorithm produced a clique including more than 11 nodes the processing took too long to be acceptable. Unfortunately, for more complicated diagrams and domains, clique sizes greater than 11 were quite common.

Fortunately, we found loopy BP to be quite successful for our task. Although the algorithm is not guaranteed to converge to correct values, we found that on our data the algorithm almost always converged. There were probably only two or three instances in hundreds of tests where the values did not converge. We initialized the messages to 1 and ran the algorithm until node values were stable to within 0.001.

Loopy BP was significantly faster than the junction tree algorithm, but for complex data it was still occasionally slower than we wished. To speed up the system's performance, we added two restrictions. First, we terminated the belief propagation algorithm after 60 seconds of processing if it had not converged by this time. This restriction was needed only about a dozen times in the 80 circuit diagrams we processed, but it prevented the rare case where belief propagation took 20 minutes to converge. Second, we allowed each node to have no more than eight parents (i.e., only eight higher-level hypotheses could be considered for a single hypothesis). This restriction ensured a limit on the complexity of the graphs produced by the system. For the family tree domain, this limitation had no effect on the system's performance because the system never generated more than eight higher-level hypotheses for a lower-level hypothesis. However, in the circuit domain, higher-level hypotheses were occasionally prevented from being considered due to this limitation. For complex domains such as circuit diagrams, we will need to work on finding more efficient inference algorithms to allow the system to process more complex networks in a reasonable amount of time. We will also explore other methods of simplifying the network structure that do not prevent the system from considering possibly correct hypotheses.

## 2.6 Hypothesis Generation

The major challenge in hypothesis generation is to generate the correct interpretation as a candidate hypothesis without generating too many to consider in real-time.

Our method of evaluating partial interpretations allows us to use a bottom–up/top–down generation strategy that greatly reduces the number of hypotheses considered but still generates the correct interpretation for most shapes in the sketch.

Our hypothesis generation algorithm has three steps.

1. Bottom–up step: As the user draws, the system parses the strokes into primitive objects using a domain-independent recognition toolkit developed in previous work [48]. Compound interpretations are hypothesized for each compound object that includes these low-level shapes, even if not all the subshapes of the pattern have been found.
2. Top–down step: The system attempts to find subshapes that are missing from the partial interpretations generated in step 1, often by reinterpreting strokes that are temporally and spatially proximal to the proposed shape.
3. Pruning step: The system removes unlikely interpretations.

This algorithm, together with the Bayesian network representation presented above, deals successfully with the challenges presented in Sect. 2.2. Using the example in Fig. 2.13, we illustrate how the system generates hypotheses that allow the Bayesian network mechanism to resolve noise and inherent ambiguity in the sketch, how the system manages the number of potential interpretations for the sketch, how the system recovers from low-level recognition errors, and how the system allows for variation in drawing style. For a more detailed description of how we handle specific challenges in hypothesis generation, see [1].

Based on low-level interpretations of a stroke, the bottom–up step generates a set of hypotheses to be evaluated using the Bayesian network mechanism presented in the previous section. In the sketch in Fig. 2.13, the user's first stroke is correctly identified as an ellipse by the low-level recognizer, and from that ellipse the system generates the interpretation `ellipse`, and in turn, partial interpretations (templates) for mother-son, mother-daughter, father-daughter, marriage, partner-female, and divorce. These proposed interpretations have empty slots into which future interpretations will be filled in.

Naive bottom–up interpretation easily can generate too many hypotheses to consider in real-time. We employ three strategies to control the number of hypotheses generated in the bottom–up step. First, when an interpretation can be fit into more than one slot in a higher-level template (e.g., in Fig. 2.14, L1 could be the shaft or either of the lines in the head of A1), the system arbitrarily chooses one of the valid slots rather than generating one hypothesis for each potential fit. Later, the system can shuffle the shapes in the template when it attempts to fit more subshapes.

Second, the system does not generate higher-level interpretations for interpretations that are only partially filled. The lines generated from Strokes 4 and 5 in Fig. 2.13 result in one partial hypothesis—`arrow` (A1)—and two complete hypotheses—`quadrilateral` (Q1) and `marriage-link` (ML1) (Fig. 2.14). Continuing to generate higher-level templates from partial hypotheses would yield a large number of hypotheses (one hypothesis for each higher-level domain pattern involving each existing partial hypothesis). To avoid this explosion, the system continues to generate templates using only the complete hypotheses (in this case, ML1 and Q1).

Third, when the system processes polylines, it assumes that all the lines in a single polyline will be used in one interpretation. While this assumption does not always hold, in practice we find that it is often true and greatly reduces the number of possible interpretations. The system recognizes Stroke 2 as a four-line polyline. The bottom–up step generates only a quadrilateral because that is the only shape in the domain that requires four lines.

The top–down step allows our system to recover from low-level recognition errors. Stroke 3 is incorrectly, but reasonably, parsed into five lines by the low-level recognizer. Because the system does not know about any five-line objects, but does know about things that contain fewer than five lines, it attempts to re-segment the stroke into two lines, three lines and four lines (with a threshold on acceptable error). It succeeds in re-segmenting the stroke into four lines and successfully recognizes the lines as a quadrilateral. Although the four-line fit is not perfect, the network allows the context of the quadrilateral in addition to the stroke data to influence the system's belief in the four-line interpretation. Also note that the five lines from the original segmentation remain in the interpretation network.

The system controls the number of interpretations in the network through pruning, which occasionally causes it to prune a correct hypothesis before it is complete. The top–down step regenerates previously pruned hypotheses, allowing the system to correctly interpret a symbol despite variations in drawing order. The left-most arrow in Fig. 2.2 was drawn with two non-consecutive strokes (Strokes 8 and 12). In response to Stroke 8, the system generates both an arrow partial hypothesis and a marriage-link hypothesis (using the line hypothesis generated for this stroke). Because the user does not immediately complete the arrow, and because the competing marriage-link hypothesis is complete and has a high probability, the system prunes the arrow hypothesis after Stroke 9 is drawn. Later, Stroke 12 is interpreted as a two-line polyline and a new arrow partial hypothesis is generated. The top–down step then completes this arrow interpretation using the line generated previously from Stroke 8, effectively regenerating a previously pruned interpretation.

### *2.6.1 Selecting an Interpretation*

As each stroke is drawn, the sketch system uses a greedy algorithm to select the best interpretation for the sketch. It queries the Bayesian network for the strongest complete interpretation, sets aside all the interpretations inconsistent with this choice, chooses the next most likely remaining domain interpretation, and so forth. It leaves strokes that are part of partial hypotheses uninterpreted. Although the system selects the most likely interpretation at every stroke, it does not eliminate other interpretations. Partial interpretations remain and can be completed with the user's subsequent strokes. Additionally, the system can change its interpretation of a stroke when more context is added.

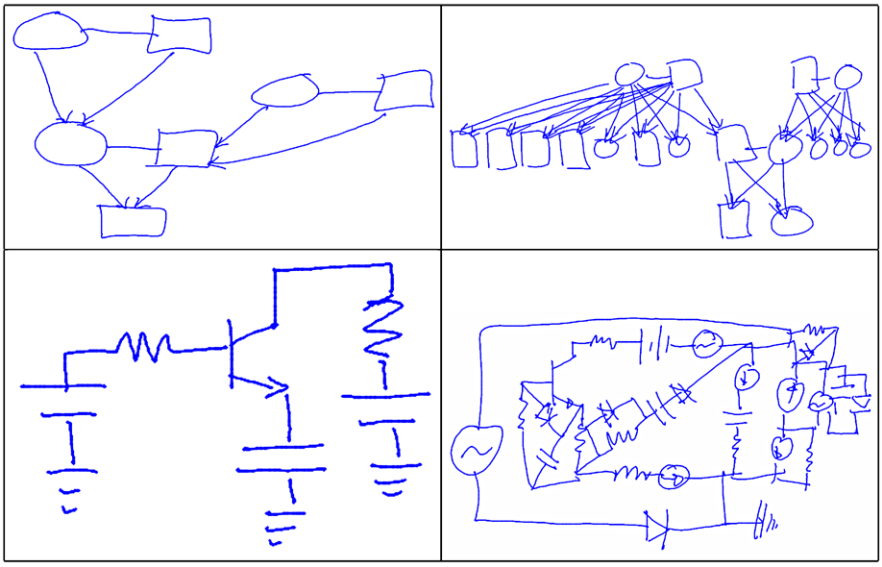


**Fig. 2.15** Examples that illustrate the range of complexity of the sketches collected

## 2.7 Application and Results

Applying our complete system, called SketchREAD (Sketch Recognition Engine for mAny Domains), to a particular domain involves two steps: specifying the structural descriptions for the shapes in the domain and specifying the prior probabilities for the domain patterns and any top-level shapes (i.e., those not used in domain patterns, which, consequently, will not have parents in the generated Bayesian network. See [1] for details on how probabilities are assigned to other shapes). We applied SketchREAD to two domains: family trees and circuits. For each domain, we wrote a description for each shape and pattern in that domain and estimated the necessary prior probabilities by hand. Through experimentation, we found the recognition performance to be insensitive to the exact values of these priors.

We ran SketchREAD on a set of ten family tree diagrams and 80 circuit diagrams we collected from users.[3] Examples of these sketches are given in Fig. 2.15. We present qualitative results, as well as aggregate recognition and running time results for each domain. Our results illustrate the complexity our system can currently handle, as well as the system's current limitations. We discuss those limitations below, describing how best to use the system in its current state and highlighting what needs to be done to make the system more powerful. Note that to apply the system to each domain, we simply loaded the domain's shape information; we did not modify the recognition system.

[3]To collect these sketches we asked users to perform synthesis tasks (i.e. not to copy pre-existing diagrams) and performed no recognition while they were sketching.

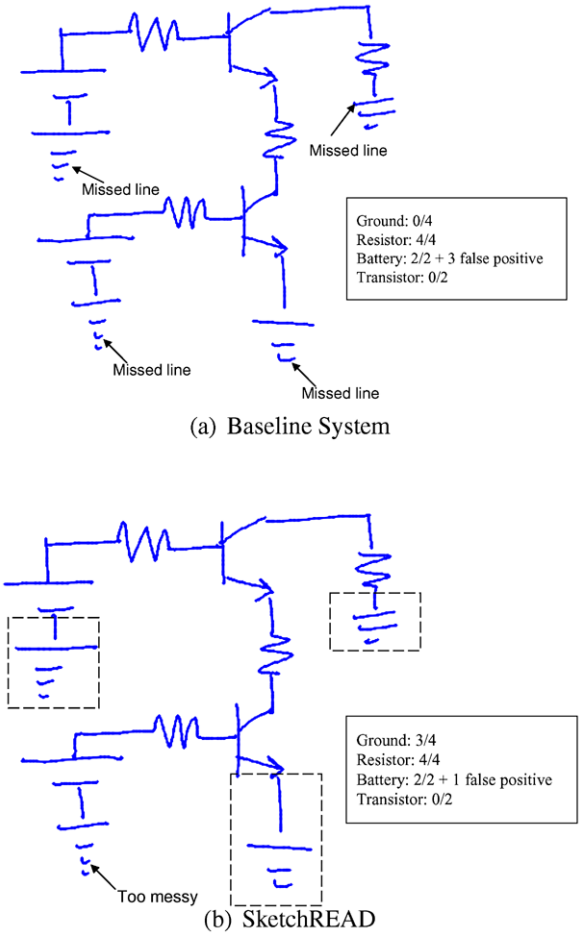



**Fig. 2.16** Recognition performance example. Overall recognition results (# correct/total) are shown in the boxes

Figure 2.16 illustrates how our system is capable of handling noise in the sketch and recovering from missed low-level interpretations. In the baseline case, one line from each ground symbol was incorrectly interpreted at the low-level, causing the ground interpretations to fail. SketchREAD was able to reinterpret those lines using the context of the ground symbol in three of the four cases to correctly identify the symbol. In the fourth case, one of the lines was simply too messy, and SketchREAD preferred to (incorrectly) recognize the top two lines of the ground symbol as a battery.

In evaluating our system's performance, direct comparisons with previous work are difficult, as there are few (if any) published results for this type of recognition task, and those that are published are tested on different (unavailable) datasets. We compared SketchREAD's recognition performance with the performance of a strictly bottom–up approach of the sort used in previous systems [2, 42]. This strictly bottom–up approach combined low-level shapes into higher-level patterns without

**Table 2.1** Recognition rates for the baseline system (BL) and SketchREAD (SR) for each sketch for the family tree domain. The size column indicates the number of strokes in each sketch

| | Size | #Shapes | % Correct | |
|---|---|---|---|---|
| | | | BL | SR |
| Mean | 50 | 34 | 50 | 77 |
| S1 | 24 | 16 | 75 | 100 |
| S2 | 28 | 16 | 75 | 87 |
| S3 | 29 | 23 | 57 | 78 |
| S4 | 32 | 22 | 31 | 81 |
| S5 | 38 | 31 | 54 | 87 |
| S6 | 48 | 36 | 58 | 78 |
| S7 | 51 | 43 | 26 | 72 |
| S8 | 64 | 43 | 49 | 74 |
| S9 | 84 | 49 | 42 | 61 |
| S10 | 102 | 60 | 57 | 80 |

top–down reinterpretation. Even though our baseline system did not reinterpret low-level interpretations, it was not trivial. It could handle some ambiguities in the drawing (e.g., whether a line should be interpreted as a marriage-link or the side of a quadrilateral) using contextual information in the bottom–up direction. To encourage others to compare their results with those presented here we have made our test set publicly available at http://rationale.csail.mit.edu/ETCHASketches.

We measured recognition performance for each system by determining the number of correctly identified objects in each sketch (Tables 2.1 and 2.2). For the family tree diagrams SketchREAD performed consistently and notably better than our baseline system. On average, the baseline system correctly identified 50% of the symbols, while SketchREAD correctly identified 77%, a 54% reduction in the number of recognition errors. Due to inaccurate low-level recognition, the baseline system performed quite poorly on some sketches. Improving low-level recognition would improve recognition results for both systems; however, SketchREAD reduced the error rate by approximately 50% independent of the performance of the baseline system. Because it is impossible to build a perfect low-level recognizer, SketchREAD's ability to correct low-level errors will always be important.

Circuit diagrams present SketchREAD with more of a challenge for several reasons. First, there are more shapes in the circuit diagram domain and these shapes are more complex. Second, there is a stronger degree of overlap between shapes in the circuit diagrams. For example, it can be difficult to distinguish between a capacitor and a battery. As another example, a ground symbol contains within it (at least one) battery symbol. Finally, there is more variation in the way people draw circuit diagrams, and their sketches are messier causing the low-level recognizer to fail more often. They tend to include more spurious lines and over-tracings.

Overall, SketchREAD correctly identified 62% of the shapes in the circuit diagrams, a 17% reduction in error over the baseline system. It was unable to handle more complex shapes, such as transistors, because it often failed to generate the

**Table 2.2** Aggregate recognition rates for the baseline system (BL) and SketchREAD (SR) for the circuit diagrams by shape

| | Total | % Correct | | # False Pos | |
|---|---|---|---|---|---|
| | | BL | SR | BL | SR |
| AC Source | 4 | 100 | 100 | 35 | 29 |
| Battery | 96 | 60 | 89 | 56 | 71 |
| Capacitor | 39 | 56 | 69 | 27 | 14 |
| Wire | 1182 | 62 | 67 | 478 | 372 |
| Ground | 98 | 18 | 55 | 0 | 5 |
| Resistor | 330 | 51 | 53 | 7 | 8 |
| Voltage Src. | 43 | 2 | 47 | 1 | 8 |
| Diode | 77 | 22 | 17 | 0 | 0 |
| Current Src. | 44 | 7 | 16 | 0 | 0 |
| Transistor | 43 | 0 | 7 | 0 | 14 |

correct mapping between strokes and pieces of the template. Although the system attempts to shuffle subshapes in a template in response to new input, for the sake of time it cannot consider all possible mappings of strokes to templates. We discuss below how we might extend SketchREAD to improve its performance on complex domains such as circuit diagrams.

We measured SketchREAD's running time to determine how it scales with the number of strokes in the sketch. Figure 2.17 graphs the median time to process each stroke for each domain. The vertical bars in the graph show the standard deviation in processing time over the sketches in each domain. (One family tree diagram took a particularly long time to process because of the complexity of its interpretation network, discussed below. This sketch affected the median processing time only slightly but dominated the standard deviation. It has been omitted from the graph for clarity.) Three things about these graphs are important. First, although SketchREAD does not yet run in real-time, the time to process each stroke in general increased only slightly as the sketch got larger. Second, not every stroke was processed by the system in the same amount of time. Finally, the processing time for the circuit diagrams is longer than the processing time for the family trees.

By instrumenting the system, we determined that the processing time is dominated by the inference in the Bayesian network, and all of the above phenomena can be explained by examining the size and complexity of the interpretation network. The number of nodes in the interpretation network grows approximately linearly as the number of strokes increases. This result is encouraging, as the network would grow exponentially using a naïve approach to hypothesis generation. The increase in graph size accounts for the slight increase in processing time in both graphs. The spikes in the graphs can be explained by the fact that some strokes not only increased the size of the network, but had more higher-level interpretations, creating more fully connected graph structures, which causes an exponential increase in inference time. After being evaluated, most of these high-level hypotheses were immediately pruned, accounting for the sharp drop in processing time on the next

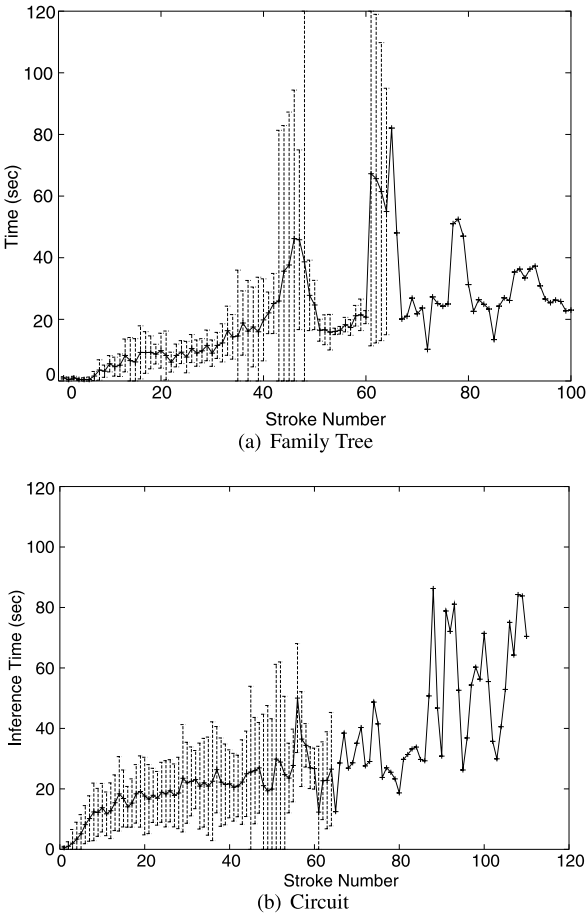



**Fig. 2.17** The median incremental time it took the system to process each stroke in the family tree and circuit diagrams. *Vertical bars* show the standard deviation across the sketches in each domain

(a) Family Tree

(b) Circuit

stroke. Finally, the fact that circuits take longer to process than family trees is related to the relative complexity of the shapes in the domain. There are more shapes in the circuit diagram domain and they are more complex, so the system must consider more interpretations for the user's strokes, resulting in larger and more connected Bayesian networks.

## 2.8 Remaining Challenges and Extensions

SketchREAD significantly improves the recognition performance of unconstrained sketches. However, its accuracy, especially for complicated sketches and domains, is still too low to be practical in most cases. Here we consider how to improve the system's performance, and in particular, describe a promising method for aiding with segmentation, without placing constraints on the users' drawing style.

First, while SketchREAD always corrected some low-level interpretation errors, its overall performance still depended on the quality of the low-level recognition. Our low-level recognizer was highly variable and could not cope with some users' drawing styles. In particular, it often missed corners of polylines, particularly for symbols such as resistors. Recently proposed, more accurate corner-finding techniques [60] will help address these problems.

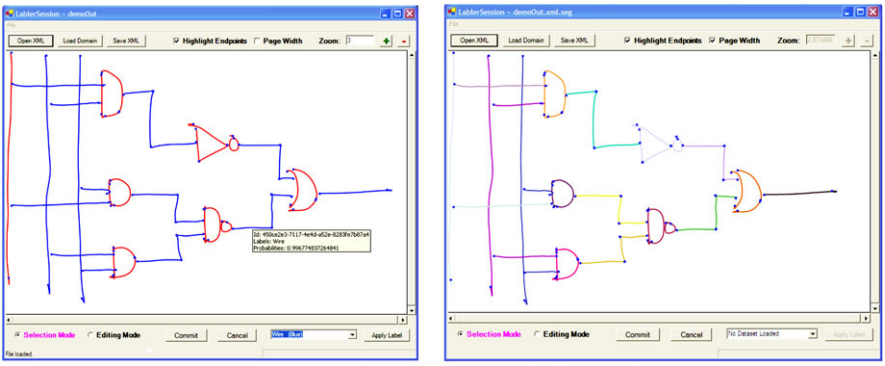
Second, although in general SketchREAD's processing time scaled well as the number of strokes increased, it occasionally ran for a long period. The system had particular trouble with areas of the sketch that involved many strokes drawn close together in time and space and with domains that involve more complicated or overlapping symbols. This increase in processing time was due almost entirely to in increase in Bayesian network complexity.

We suggest two possible solutions. First, part of the complexity arises because the system tries to combine new strokes with low-level interpretations to form correct high-level interpretations (e.g., the four lines that make a quadrilateral). These new interpretations were pruned immediately, but they increased the size and complexity of the network temporarily, causing the bottlenecks noted above. In response, we are testing methods for "confirming" older interpretations and removing their subparts from consideration other higher-level interpretations as well as confirming their values in the Bayesian network so that their posterior probabilities do not have to be constantly re-computed. Second, we can modify the belief propagation algorithm we are using. We currently use Loopy Belief Propagation, which repeatedly sends messages between the nodes until each node has reached a stable value. Each time the system evaluates the graph, it resets the initial messages to one, essentially erasing the work that was done the last time inference was performed, even though most of the graph remains largely unchanged. Instead, this algorithm should begin by passing the messages it passed at the end of the previous inference step.

Third, because our recognition algorithm is stroke-based, spurious lines and overtracing hindered the system's performance in both accuracy and running time. A preprocessing step to merge strokes into single lines would likely greatly improve the system's performance. Also, in the circuit diagram domain, users often drew more than one object with a single stroke. A preprocessing step could help the system segment strokes into individual objects.

### *2.8.1 Using Single-Stroke Classification to Improve Grouping*

Many of the above problems were caused by the difficulty of performing simultaneous segmentation and recognition. In SketchREAD, recognition and segmentation are inherently intertwined: the various hypotheses dictate different stroke segmentations. Using our template-based recognition approach to dictate segmentation means that our system cannot rely on segmentation to limit the number of possible interpretations, nor can it apply vision-based algorithms efficiently to sets of strokes known to comprise a single object. However, as discussed in Sect. 2.2, there is no



(a) Sketch after stroke-level recognition (b) Sketch after stroke grouping

**Fig. 2.18** Single-stroke recognition and grouping

reliable purely spatial or temporal method to segment strokes into individual objects in freely-drawn sketches.

Recently, researchers have developed a technique to roughly classify single strokes, and this classification be used to inform the process of sketch segmentation. The technique relies on the fact that strokes can be roughly grouped into categories individually by looking at their local properties and their relationships to other strokes in the diagram. For example, in the circuit diagram in Fig. 2.18 the strokes that make up the gates tend to be shorter and have a higher curvature than the strokes that make up the wires. The wire strokes and gate strokes also have a well-defined relationship to one another. Then, once strokes individually are classified as either wires or gates (Fig. 2.18(a)) they are sufficiently separated in both time and space that simple clustering algorithms can successfully group strokes into individual objects (Fig. 2.18(b)), which can then be recognized by any number of sketch recognition algorithms, including the Bayesian network approach presented in this chapter.

Szummer and Qi [53] developed a method for classifying individual strokes based on local and contextual information based using conditional random fields. They illustrate its success on organizational chart diagrams. We have applied their approach to the more complex domain of circuit diagrams and find that it performs quite well.

Briefly, a CRF is an undirected graphical model that represents the conditional probability distribution $P(\mathbf{y} \mid \mathbf{x})$ where $\mathbf{x}$ is a set of input data and $\mathbf{y}$ is a set of labels for these data. The actual CRF consists of a graph $G = (V, E)$ and an associated set of potential functions that together define $P(\mathbf{y} \mid \mathbf{x})$. Each node in $V$ corresponds to an element to label (i.e. the members of $\mathbf{y}$), and each edge in $E$ quantifies a probabilistic dependence between these elements. For more details, see [53].

In our application, the vector $\mathbf{x}$ represents the stroke data while the vector $\mathbf{y}$ represents the labels for each stroke. $P(\mathbf{y} \mid \mathbf{x})$, then, is simply the probability of a given labeling for each stroke, given properties of the stroke data.

We automatically create the graph by creating a node for each stroke and linking nodes for strokes that are spatially or temporally proximal. We find that for some domains, including the digital circuit domain, fragmenting strokes at their corners before creating the graph, as in [53], degrades performance. We consider two types of potential functions: site potentials that measure the compatibility between a stroke and its associated label, and pairwise interaction potentials that measure the compatibility between neighboring labels. Both types of potentials measure compatibility by linearly combining parameters with a set of feature functions and passing the result through a non-linearity (we use the exponential).

After labeling each stroke, we group strokes into individual objects. Even with perfect labels, stroke grouping is not trivial. For example, different wires may overlap in space and the same wire may be separated in time. We use a graph theoretic method for stroke grouping that treats each labeled stroke as a node in a graph, with edges between adjacent strokes. The algorithm then finds the connected components in the graph.

Two strokes are adjacent if their minimum distance is lower than a given threshold. We designed specific distance metrics for the digital circuit domain. Given two strokes, if neither stroke is a wire, then the minimum distance between the strokes is the minimum distance between any two points in the strokes. If either stroke is a wire, the minimum distance between the strokes is the distance from an endpoint to any other point on the other stroke. We use this modified distance because wires frequently overlap even when they are not meant to represent the same component. In both cases, the minimum distance is normalized by the sum of the diagonals of the smallest bounding box around each of the strokes. This normalization provides a unitless measure that is invariant under uniform scaling.

Although this work is still in progress, our initial results in this area are promising. We tested our CRF for single-stroke classification on digital circuit diagrams, classifying strokes as text, wires or gates, and achieve 93% overall accuracy and 77% accuracy in stroke segmentation. Figure 2.18 shows one example result.

## 2.9 Conclusion

This chapter has presented an approach to multi-domain sketch recognition using dynamically constructed Bayesian networks. We have shown how to use context to improve online sketch interpretation and demonstrated its performance in SketchREAD, an implemented sketch recognition system that can be applied to multiple domains. We have shown that SketchREAD is more robust and powerful than previous systems at recognizing unconstrained sketch input in a domain. The capabilities of this system have applications both in human computer interaction and artificial intelligence. Using and building on this approach, we will be able to explore further the nature of usable intelligent computer-based sketch systems and gain a better understanding of what people would like from a drawing system that is capable of understanding their freely-drawn sketches as more than just strokes. This work

provides a necessary step in uniting artificial intelligence technology with novel interaction technology to make interacting with computers more like interacting with humans.

**Acknowledgements** This work is based on my PhD thesis, supervised by Randall Davis at the Massachusetts Institute of Technology. Recent work is funded by an NSF CAREER award (IIS-0546809).

## References

1. Alvarado, C.: Multi-domain sketch understanding. PhD thesis, MIT (2004)
2. Alvarado, C., Davis, R.: Resolving ambiguities to create a natural sketch based interface. In: Proceedings of IJCAI-2001 (2001)
3. Alvarado, C., Davis, R.: Sketchread: A multi-domain sketch recognition engine. In: Proc. UIST (2004)
4. Alvarado, C., Davis, R.: Dynamically constructed Bayes nets for sketch understanding. In: Proceedings of IJCAI '05 (2005)
5. Alvarado, C., Lazzareschi, M.: Properties of real-world digital logic diagrams. In: Proc. of the 1st International Workshop on Pen-Based Learning Technologies (PLT-07) (2007)
6. Bishop, C.M., Svensen, M., Hinton, G.E.: Distinguishing text from graphics in on-line handwritten ink. In: IWFHR '04: Proceedings of the Ninth International Workshop on Frontiers in Handwriting Recognition, pp. 142–147. IEEE Computer Society, Washington (2004)
7. Blostein, D., Haken, L.: Using diagram generation software to improve diagram recognition: A case study of music notation. IEEE Transactions on Pattern Analysis and Machine Intelligence **21**(11) (1999)
8. Buxton, B.: Sketching User Experiences: Getting the Design Right and the Right Design. Morgan Kaufmann, San Mateo (2007)
9. Caetano, A., Goulart, N., Fonseca, M., Jorge, J.: Sketching user interfaces with visual patterns. In: Proceedings of the 1st Ibero-American Symposium in Computer Graphics (SIACG02), pp. 271–279 (2002)
10. Charniak, E.: Bayesian networks without tears: making Bayesian networks more accessible to the probabilistically unsophisticated. Artificial Intelligence **12**(4), 50–63 (1991)
11. Cohen, P.R., Johnston, M., McGee, D., Oviatt, S., Pittman, J., Smith, I., Chen, L., Clow, J.: Quickset: Multimodal interaction for distributed applications. In: ACM Multimedia'97, pp. 31–40. ACM Press, New York (1997)
12. Do, E.Y.L., Gross, M.D.: Drawing as a means to design reasoning. AI and Design (1996)
13. Forbus, K.D., Usher, J., Chapman, V.: Sketching for military course of action diagrams. In: Proceedings of IUI (2003)
14. Forsberg, A.S., Dieterich, M.K., Zeleznik, R.C.: The music notepad. In: Proceedings of UIST '98. ACM SIGGRAPH. ACM, New York (1998)
15. Futrelle, R.P., Nikolakis, N.: Efficient analysis of complex diagrams using constraint-based parsing. In: ICDAR-95 (International Conference on Document Analysis and Recognition), Montreal, Canada, pp. 782–790 (1995)
16. Gennari, L., Kara, L.B., Stahovich, T.F.: Combining geometry and domain knowledge to interpret hand-drawn diagrams. Computers and Graphics: Special Issue on Pen-Based User Interfaces (2005)
17. Getoor, L., Friedman, N., Koller, D., Pfeffer, A.: Learning probabilistic relational models. In: IJCAI, pp. 1300–1309 (1999). http://citeseer.nj.nec.com/friedman99learning.html
18. Glessner, S., Koller, D.: Constructing flexible dynamic belief networks from first-order probabilistinc knowledge bases. In: Symbolic and Quantitative Approaches to Reasoning and Uncertainty, pp. 217–226 (1995)

19. Goldman, R.P., Charniak, E.: A language for construction of belief networks. IEEE Transactions on Pattern Analysis and Machine Intelligence **13**(3) (1993)
20. Grimson, W.E.L.: The combinatorics of heuristic search termination for object recognition in cluttered environments. IEEE Transactions on PAMI **13**(9), 920–935 (1991)
21. Gross, M.D.: The electronic cocktail napkin—a computational environment for working with design diagrams. Design Studies **17**, 53–69 (1996)
22. Gross, M., Do, E.Y.L.: Ambiguous intentions: A paper-like interface for creative design. In: Proceedings of UIST 96, pp. 183–192 (1996)
23. Haddawy, P.: Generating Bayesian networks from probability logic knowledge bases. In: Proceedings of UAI '94 (1994)
24. Hammond, T., Davis, R.: Tahuti: A geometrical sketch recognition system for UML class diagrams. In: AAAI Spring Symposium on Sketch Understanding, 59–68 (2002)
25. Hammond, T., Davis, R.: LADDER: A language to describe drawing, display, and editing in sketch recognition. In: Proceedings of the 2003 International Joint Conference on Artificial Intelligence (IJCAI) (2003)
26. Hammond, T., Davis, R.: Automatically transforming symbolic shape descriptions for use in sketch recognition. In: Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-04) (2004)
27. Hammond, T., Davis, R.: Interactive learning of structural shape descriptions from automatically generated near-miss examples. In: IUI '06: Proceedings of the 11th International Conference on Intelligent User Interfaces, pp. 210–217. ACM, New York (2006)
28. Hse, H., Newton, A.R.: Recognition and beautification of multi-stroke symbols in digital ink. Computers and Graphics (2005)
29. Jensen, F.V., Lauritzen, S.L., Olesen, K.G.: Bayesian updating in causal probabilistic networks by local computations. Computational Statistics Quarterly **4**, 269–282 (1990)
30. Jensen, F.V.: Bayesian Networks and Decision Graphs. Statistics for Engineering and Information Science. Springer, Berlin (2001)
31. Kara, L.B., Stahovich, T.F.: Hierarchical parsing and recognition of hand-sketched diagrams. In: Proc. of UIST '04 (2004)
32. Koller, D., Pfeffer, A.: Object-oriented Bayesian networks. In: Proceedings of the Thirteenth Annual Conference on Uncertainty, Providence, RI, pp. 302–313 (1997)
33. Labahn, G., MacLean, S., Marzouk, M., Rutherford, I., Tausky, D.: Mathbrush: An experimental pen-based math system. In: Dagstuhl Seminar Proceedings, Challenges in Symbolic Computation Software (2006)
34. Landay, J.A., Myers, B.A.: Interactive sketching for the early stages of user interface design. In: Proceedings of CHI '95: Human Factors in Computing Systems, pp. 43–50 (1995)
35. Lank, E.H.: A retargetable framework for interactive diagram recognition. In: Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR'03) (2003)
36. Lank, E., Thorley, J.S., Chen, S.J.S.: An interactive system for recognizing hand drawn UML diagrams. In: Proceedings for CASCON (2000)
37. Laskey, K.B., Mahoney, S.M.: Network fragments: Representing knowledge for constructing probabilistic models. In: Proceedings of UAI '97 (1997)
38. Lauritzen, S.L., Spiegelhalter, D.J.: Local computations with probabilities on graphical structures and their application to expert systems. Journal of the Royal Statistical Society **50**(2), 157–224 (1988)
39. LaViola, J., Zeleznik, R.: Mathpad2: A system for the creation and exploration of mathematical sketches. ACM Transactions on Graphics (Proceedings of SIGGRAPH 2004) **23**(3) (2004)
40. Lu, W., Wu, W., Sakauchi, M.: A drawing recognition system with rule acquisition ability. In: Proceedings of the Third International Conference on Document Analysis and Recognition, vol. 1, pp. 512–515 (1995)
41. Matsakis, N.: Recognition of handwritten mathematical expressions. Master's thesis, Massachusetts Institute of Technology (1999)

42. Newman, M.W., Lin, J., Hong, J.I., Landay, J.A.: DENIM: An informal Web site design tool inspired by observations of practice. Human-Computer Interaction **18**(3), 259–324 (2003)
43. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Mateo (1988)
44. Pfeffer, A., Koller, D., Milch, B., Takusagawa, K.: SPOOK: A system for probabilistic object-oriented knowledge representation. In: Proceedings of UAI '99, pp. 541–550 (1999)
45. Poole, D.: Probabilistic horn abduction and Bayesian networks. Artificial Intelligence (1993)
46. Saund, E., Fleet, D., Larner, D., Mahoney, J.: Perceptually supported image editing of text and graphics. In: Proceedings of UIST '03 (2003)
47. Sezgin, T.M., Davis, R.: Sketch interpretation using multiscale models of temporal patterns. IEEE Computer Graphics and Applications **27**(1), 28–37 (2007). doi:10.1109/MCG.2007.17
48. Sezgin, T.M., Stahovich, T., Davis, R.: Sketch based interfaces: Early processing for sketch understanding. In: The Proceedings of 2001 Perceptive User Interfaces Workshop (PUI'01), Orlando, FL (2001)
49. Shilman, M., Pasula, H., Russell, S., Newton, R.: Statistical visual language models for ink parsing. In: Sketch Understanding, Papers from the 2002 AAAI Spring Symposium, pp. 126–132. AAAI Press, Stanford (2002)
50. Shilman, M., Viola, P., Chellapilla, K.: Recognition and grouping of handwritten text in diagrams and equations. In: Proceedings of the International Workshop on Frontiers in Handwriting Recognition (IWFHR) (2004)
51. Stahovich, T., Davis, R., Shrobe, H.: Generating multiple new designs from a sketch. Artificial Intelligence **104**(1–2), 211–264 (1998)
52. Strat, T.M., Fischler, M.A.: Context-based vision: Recognizing objects using information from both 2-d and 3-d imagery. IEEE Transactions on Pattern Analysis and Machine Intelligence **13**(10), 1050–1065 (1991)
53. Szummer, M., Qi, Y.: Contextual recognition of hand-drawn diagrams with conditional random fields. In: Proceedings of the 9th Int. Workshop on Frontiers in Handwriting Recognition (IWFHR), pp. 32–37 (2004)
54. Tenneson, D.: Technical report on the design and algorithms of chempad. Technical report, Brown University (2005)
55. Torralba, A., Sinha, P.: Statistical context priming for object detection. In: Proceedings of ICCV '01, pp. 763–770 (2001)
56. Ullman, D.G., Wood, S., Craig, D.: The importance of drawing in the mechanical design process. Computers and Graphics **14**(2), 263–274 (1990)
57. Veselova, O., Davis, R.: Perceptually based learning of shape descriptions. In: Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-04) (2004)
58. Wang, X., Biswas, M., Raghupathy, S.: Addressing class distribution issues of the drawing vs writing classification in an ink stroke sequence. In: SBIM '07: Proceedings of the 4th Eurographics Workshop on Sketch-based Interfaces and Modeling, pp. 139–146. ACM, New York (2007). doi:10.1145/1384429.1384458
59. Weiss, Y.: Belief propagation and revision in networks with loops. Technical report, AI Memo No. 1616, CBCL Paper No. 155, Massachusetts Institute of Technology (1997)
60. Wolin, A., Hammond, T.: Shortstraw: A simple and effective corner finder for polylines. In: Alvarado, C., Cani, M.P. (eds.) Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM) (2008)
61. Bayesian network tools in java (bnj). http://bnj.sourceforge.net