New Investigators: Artificial Intelligence

Location: TBA

Simplifying Sketch Recognition UI Development

Tracy A. Hammond (Texas A&M University)

Tracy Hammond is an Assistant Professor at Texas A&M University. She earned her B.A. in math, B.S. in applied math, M.S. in computer science, and M.A. in anthropology from Columbia University, as well as her Ph.D. in computer science from the Massachusetts Institute of Technology. She was an instructor at Columbia University where she taught a number of courses to undergrads and gifted high school students. For four years, she was a telecom analyst at Goldman Sachs, where she designed, developed, implemented and administered global computer telephony applications.

Abstract: Sketch recognition systems are time-consuming to build and require signal-processing expertise if they are to handle the intricacies of each domain. Our goal is to enable user interface designers and domain experts, who may not have expertise in sketch recognition, to be able to build these sketch systems. We created and implemented a new framework: FLUID (facilitating LADDER-based user interface development), in which developers can specify how domain shapes are to be identified, displayed, and edited either by drawing an example shape or typing a LADDER (a perceptually-based language to describe how shapes are drawn, displayed, and edited in a sketch recognition UI) descriptions. The GUILD (generator of user interfaces using LADDER descriptions) automatically generates a sketch recognition user interface for that domain, greatly reducing the time and expertise needed to create a new sketch interface.

Continuing in pursuit of our goal to facilitate UI development, we noted that 1) human generated descriptions contained syntactic and conceptual errors, and that 2) it is more natural for a user to specify a shape by drawing it than by editing text. However, computer generated descriptions from a single drawn example are also flawed, as one cannot express all allowable variations in a single example. In response, we created a modification of the traditional model of active learning in which the system selectively generates its own near-miss examples and uses the human teacher as a source of labels. System generated near-misses offer a number of advantages. Human generated examples are tedious to create and may not expose problems in the current concept. It seems most effective for the near-miss examples to be generated by whichever learning participant (teacher or student) knows better where the deficiencies lie; this will allow the concepts to be more quickly and effectively refined. When working in a closed domain such as this one, the computer learner knows exactly which conceptual uncertainties remain, and which hypotheses need to be tested and confirmed. The system uses these labeled examples to automatically build a LADDER shape description, using a modification of the version spaces algorithm that handles interrelated constraints, and which has the ability to also learn negative and disjunctive constraints.

1. Introduction

Hand-sketching of graphical diagrams is an important part of natural interaction for a variety of tasks, including classroom learning of topics such as graphs, mechanical engineering, circuit diagrams, software design, and many others. Paper sketches offer users the freedom to draw as they would naturally; for instance, users can draw objects with any number of strokes, and strokes may be drawn in any order. However, because paper sketches are static and uninterpreted, they lack computer editing features, requiring users to completely erase and redraw objects in order to move them. In an attempt to combine the freedom provided by a paper sketch with the powerful editing and processing capabilities of an interpreted diagram, sketch recognition systems have been developed for many domains, including Java GUI...
Sketch interfaces are valuable additions to natural human-computer interactions, but developing sketch interfaces requires substantial effort. They can be quite time-consuming to build if they are to handle the intricacies of each domain. Moreover, the sketch interface developer has to be an expert in sketch recognition. We argue that to make a user-friendly system, the designer of the system should be an expert in building user interfaces, and/or an expert in the domain itself, such as a teacher who knows her subject matter well and what sort of interface would be useful for her students. This person does not necessarily have to be an expert in sketch recognition.

To this end, we developed the FLUID framework to facilitate user interface development to enable non-experts in sketch recognition to build sketch recognition interfaces [7]. To build a sketch system, a developer would only need to describe the domain-specific information.

When constructing a user interface, the domain-specific information is able to be obtained by asking the following questions:

- What are the observable states to be recognized?
- How are these states to be recognized?
- What should happen when these states are recognized?
- How can we modify these states?

In sketch recognition user interfaces, the domain-specific information is obtained by asking these questions:

- What shapes are in the domain?
- How is each shape recognized?
- What should happen after each shape is recognized?
- How can the shapes be edited?

Many domain-specific events can occur after a shape is recognized, but what is common in most domains is a change in display. Sketchers often prefer to have a change in display to confirm that their object was recognized correctly, as a form of positive feedback. Changes in display may also function as a way to remove clutter from the diagram. For example, the system may replace several messy hand-drawn strokes with a small representative image. A change in display may vary from a simple change in color, a moderate change of cleaning up the drawn strokes (e.g., straightening lines, joining edges), to a more drastic change of replacing the strokes with an entirely different image. Because display changes are so popular and so common to most domains, we have included them in the domain specification.

This framework not only defines which shapes are in the domain and how they are to be recognized in the domain, it also emphasizes the importance of editing and display in creating an effective user interface. Developers of different domains may want the same shape to be displayed differently; the FLUID framework includes constraints and primitives in the language are based on human perception. In order to create effective multi-domain user-independent recognition, we have performed user studies testing shape description, drawing styles, and human perceptual thresholds. We have built improved lower-level (primitive-shape) and higher-level (hierarchical-shape) recognizers. Developers can describe shapes by hand-typing a description or by drawing a sample shape having the computer learn a description from near-miss shapes generated by the computer. Because we are automatically generating an entire user interface, developers can also describe how drawn shapes are to be beautified and edited in the interface. Figure 1 and Figure 2 show automatically generated sketch recognition systems for finite state machines and mechanical engineering.
To build a new sketch interface:

1. A developer writes a LADDER domain description describing information specific to each domain, including: what shapes are included in the domain, and how each shape is to be recognized, displayed (providing feedback to the user), and edited.

2. If so desired, the developer will write a Java file that functions as an interface between the existing back-end knowledge system (e.g., a CAD tool) and the recognition system, based on a supplied API. (Coding knowledge is needed to perform this optional step.)

3. The GUILD customizable recognition system translates the LADDER domain description into a shape recognizers, editors, and exhibitors.

4. The GUILD customizable recognition system now functions as a domain-specific sketch interface that recognizes, displays, and allows editing of the shapes in the domain, as specified in the domain description. It also connects via the Java interface to an existing back-end system (if optional step 2 is completed).

3. Near Miss Generation and Learning from a Single Example

LADDER shape descriptions can be automatically generated from a single drawn example, based on ideas from Veselova [11]. Figure 4 shows a description automatically generated from a single drawn example of the pound sign (a grid for a tic tac toe board) shown to the right of the figure. Shapes are described as if they were drawn perfectly (without accounting for signal noise). The recognition system includes thresholds to appropriately handle and recognize messy drawings.

Shapes are described by the components that make up each shape, and the constraints between them.

To generate the best-guess shape description, the system first lists all true constraints for that shape, which essentially acts as a recognizer for the shape drawn with no variations. We know that the correct description of the shape is a generalization of this list. The difficulty is choosing the appropriate generalization of this list. If we were to keep all of these constraints in our best-guess shape description, we would recognize only our very specific example of the shape, misrecognizing many variations as false negatives. If we make our description too general, it will allow too many variations, creating false positives. The system uses perceptual rules to create a best guess to determine which constraints are most likely to be intended by the user. For example, coincident endpoints, horizontal, vertical, equal-length vertically-aligned (sameY), and horizontally-aligned (sameX) are all perceptually important constraints.

This best-guess approximation is a plausible one, but, even with the perceptual rules, it is sometimes impossible for the computer to know exactly what variations are possible in a shape. Thus, the generated description, while syntactically correct, may still have conceptual errors: It may be under- or over-constrained. Figure 5 shows the difficulty of automatically generating a perfect description; the components line1 and line2 look the same in both the square and the arrow. The constraint perpendicular line1 line2 is true for both shapes, and any algorithm that would include the constraint for one shape would include it for the other. However, if the constraint is missing from the square definition, the square definition will be incorrect, as it is under-constrained, but, if the constraint is included in the arrow definition, it will be over-constrained and incorrect.

To solve this problem, we developed an algorithm using a novel form of active learning [13] that automatically generates its own suspected near-miss examples (rather than having the human teacher generate them), which are then classified as positive or negative by the developer. The system generates these examples to test whether components of its current concept description are necessary to the concept, or merely happened to be true of the initial example. (For example, is it necessary for both lines in the head of an arrow to be the same length, or was this accidental in the original example?) The system constantly updates its evolving version of the concept using these labeled examples to automatically learn a shape concept (in the form of a LADDER shape description) using a modification of the version spaces algorithm [14]. Our modification, which is based on mutually-exclusive perceptually-important constraints, better suits our domain in that it can learn shape descriptions with negative and disjunctive constraints. Figure 7 shows two automatically generated shapes to learn an arrow description.
4. Specifying a LADDER Description

Shape descriptions can be automatically generated from a single drawn example as described above, but LADDER descriptions can also encode other important information to aid recognition such as geometric context. Figure 8 shows the definition of a pin joint. In mechanical engineering, both a pin joint and a body are represented by a circle. Context is used to distinguish between the two of them. Figure 2 shows a recognized pin joint and circular body in the wheel of a car; the computer was able to distinguish between a pin joint and the body of the wheel using context. Figure 2 also give an example of various display and editing behaviors that are possible to define with the LADDER language.

The LADDER sketch recognition language also allows the developer to specify what should be displayed when the shapes on the screen are recognized. Display changes can provide recognition feedback to the user, beautify the diagram, as well as show application specific information computed. Application specific information may include a tic tac toe sketch recognition system stating that a player has won, while highlighting the winning row (as in Figure 3), or a finite state machine sketch recognition system highlighting the states that a given input string passes through one by one (as in Figure 1). LADDER allows specification of several different ways of displaying objects including displaying the original strokes, a cleaned or ideal version of the strokes, a drawing made of Java2D shapes, or a jpg/gif image. Editing behaviors such deletion, color changes, scaling, rotation, and others can be specified to occur when a shape is drawn, a mouse gesture occurs, or some other action occurs. Application specific code can also be written in Java to connect to a backend system. The state machine application (in Figure 1) code to highlight the current state requires about 10 lines of code, whereas the mechanical engineering (in Figure 2) code to interact with the Working Model CAD simulation program requires about 30 lines of code.

Future Work

Future work includes further improving sketch recognition technologies while simplifying UI development. We are currently working on building improved lower level recognizers, further modeling recognition off of human perception, and integrating multiple forms of context into the recognition process. We are building applications for classroom usage and hope to have them integrated into the classroom in the near future.

Conclusions

Sketch recognition systems can provide numerous benefits from time-saving, functional-aware brainstorming, and classroom active learning. However, these systems are time-consuming to build and require sketch recognition expertise. We have built GUILD to automatically generate sketch recognition user interfaces from hand-type LADDER descriptions or from those which can be automatically generated from a single example and system-generated near-miss examples. Thus far, over twenty users have built prototype sketch applications with this software, including mechanical engineering, circuit diagrams, UML class diagrams, UML case diagrams, course of action diagrams, finite state machines, flow charts, family trees, Kanji Japanese diagrams, chemical diagrams, football diagrams, basketball diagrams, text editors, musical notes, business process models, and architecture wall diagrams, just to name a few.

Acknowledgments

The authors would like to thank Randall Davis for his insight and advise on this work.

Bibliography

Conference on Artificial Intelligence (IJCAI-03), Acapulco, Mexico, 2003.


