Natural Error Correction Techniques for Sketch Recognition

by

Danica H. Chang

B.S., Electrical Engineering and Computer Science, UC Berkeley, 2011
M.S., Electrical Engineering and Computer Science, M.I.T., 2013

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science
in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

May 2013

Signature of Author: _____

Danica H. Chang
Department of Electrical Engineering and Computer Science
May 22, 2013

Certified by: _____

Randall Davis
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by: _____

Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Committee for Graduate Students

Natural Error Correction Techniques for Sketch Recognition

by Danica H. Chang

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 2013, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

Over the past few years, a plethora of tablet devices has made it very easy for users to input information by sketching as if on paper. In addition, sketch recognition systems help users convert these sketches into information that the computer understands. While lots of work has been done in developing better sketch recognizers, very little work has previously been done on how to edit the sketch once it's been drawn, whether the error is the user's or the sketch recognizer's. In response, we developed and studied intuitive methods of interacting with a sketch recognition system to correct errors made by both the recognizer and the user. The editor allows users to click and lasso to select parts of the sketch, label the selected strokes, erase by scribbling over strokes, and even overwrite errors. Letting users provide feedback to the sketch recognizer helps improve the accuracy of the sketch as well as allows the sketch recognizer's performance to improve over time.

Thesis Supervisor: Randall Davis
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I'd like to thank my advisor, Professor Randall Davis, for providing me direction, guidance and encouragement throughout my research.

The sketch recognizer, ChemInk, on which my work is built on, was developed by Tom Ouyang. The code Tom left behind was very well written and easy to use. Tom was an invaluable resource, explaining how his code worked and even after he had graduated he continued to occasionally help me debug issues that I came across. I also want to thank my other groupmates: Jeremy Scott, Ying Yin, Yale Song and Andrew Correa. They provided valuable feedback and suggestions throughout my research.

I would like to thank all the users in my user studies who provided me with great feedback and data. I would also like to thank Professor Troy Van Voorhis for answering my chemistry questions.

I also want to express my graditute to all my friends and family for their support and encouragement. I especially want to thank Carrie and Kendall for helping me realize the importance of my research.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation and Overview

For decades, the only way to interact with a computer was with a mouse and keyboard. Recently, a plethora of new input devices such as the Wii, Kinect, smart phones and tablets have afforded the ability to develop new and more natural forms of interaction. In the domain of sketches, there has been a lot of work done on sketch recognition and allowing users to input information through drawing, as if on paper. However, there is very little work done on how to edit the sketch once it's been drawn, or how to correct errors made by either the user or the sketch recognizer.

For example, suppose a chemist is trying to develop a new molecule. The fastest and most natural way for him to describe the molecule would be to sketch it. Using state-of-the-art sketch recognition tools, such as ChemInk (Figure 1.1), his sketch can be interpreted and transformed into a chemical structure description. However, if the chemist makes a mistake in his sketch, or the sketch recognition system makes an error, there needs to be a way for him to correct the sketch.

A sketch editor needs many of the same editing capabilities as a text editor. Text editors allow users to quickly and easily erase text. In order to erase with equal ease in a sketch editor, it must allow the user to erase using just the pen, without switching to a different mode. We achieve this by recognizing a scribble gesture and erasing the portion of the sketch that the scribble overlaps. Text editors also provide various

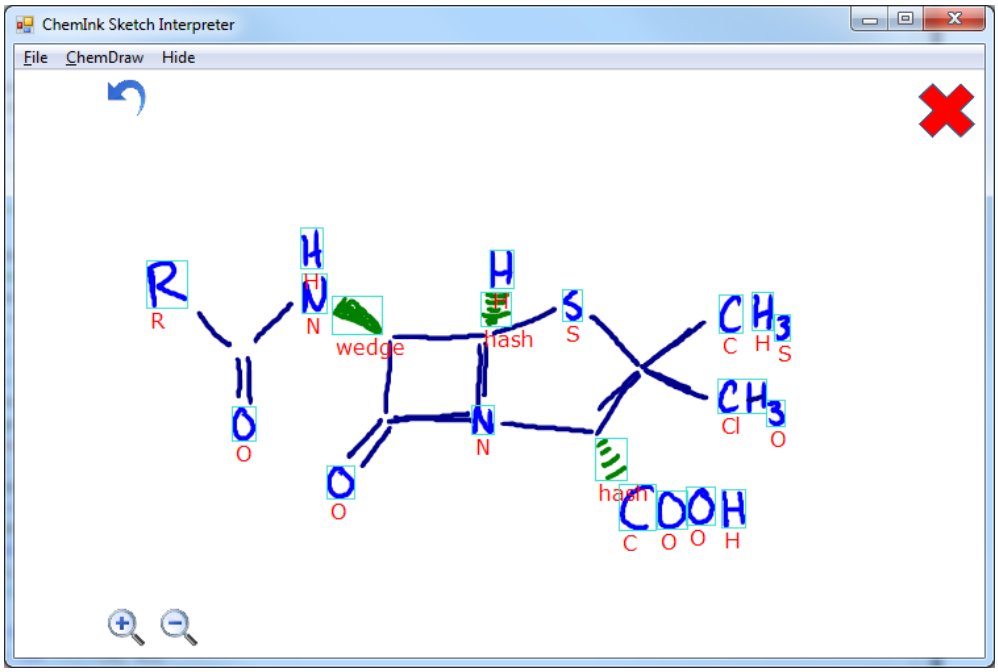


Figure 1.1: ChemInk, a chemical sketch recognition tool

methods of selecting portions of the text, such as dragging the mouse or using the arrow keys. In the sketch editor, we allow users to click or lasso portions of a sketch in order to select it.

While a sketch editor and a text editor share many of the same basic editing features, an editor for a sketch recognition system also needs to account for various recognition related edits. For example, when the sketch recognition system fails to correctly recognize a symbol, the user needs a way to communicate to the system that it has made a mistake and what the correct recognition is.

This thesis presents our work in building a sketch editor that enables natural interactions when modifying a sketch.

1.2 Contributions

This thesis makes three contributions towards the fields of human-computer interaction and intelligent user interfaces.

1. We developed a method to display the connectivity of various components of a chemical structure sketch. Users can visually see the recognition system's interpretation of not just the symbols, but also which symbols are connected to which bonds. This information is crucial for fully communicating the recognizer's understanding of the sketch and allows the user to make edits to the connectivity if any errors are present.

2. We developed and studied 4 methods for correcting misclassified portions of a sketch.

   (a) Users can select and manually label a misclassified symbol.

   (b) Users can erase and re-sketch a misclassified symbol.

   (c) Users can overwrite a misclassified symbol.

   (d) Users can tell the sketch recognizer to reattempt its recognition on a specific symbol.

3. We evaluated how natural each editing technique is through a series of user tests, asking users familiar with organic chemistry to rate the naturalness of different editing techniques.

1.3 Thesis Outline

In the following chapter, we give a brief overview of related work. Then, in Chapter 3, we will discuss display features we implemented including zooming as well as a way to visualize which bonds are connected in the underlying system. In Chapter 4, we explain

the correction features that we implemented in ChemDraw, such as lasso selection, reattempt, scribble erase, and overwrite. In Chapter 5, we describe the procedures, and experimental results of 3 user studies. Then, we summarize and conclude the thesis in Chapter 6.

Chapter 2

Related Work

2.1 Sketch Recognizer

The work for this thesis builds on past work in Multimodal Understanding Group (MUG) at MIT CSAIL. The sketch recognizer, ChemInk [6], built by Tom Ouyang allows users to sketch chemical structure diagrams on a tablet. The system recognizes and labels each part of the sketch with the corresponding symbol and can convert the entire sketch into a CAD model of the chemical. While the recognizer had a decent accuracy, it was not perfect and some symbols are labeled incorrectly due to either human error or errors in the sketch recognizer. This inspired my work to create a sketch editing system to allow users to make corrections to the labeling in the sketch. Allowing the user to provide feedback to the sketch recognizer can help create a more accurate understanding of the sketch. In addition, by using the edits as new training examples, the sketch recognizer's performance can improve in the long run.

2.2 Interacting with Recognizers

There are a variety of ways in which users can interact with a recognizer in order to correct errors. The most common techniques fall into two categories: repetition and choice [4]. Repetition involves the user correcting an interpretation by explicitly repeating some of the input. Choice allows the user to select the correct interpretation

from a set of choices presented by the recognizer.

2.3 Selections

Selecting parts of a sketch is an important capability as it allows users to indicate the portion of the sketch they want to edit. Two main pen-based selection strategies have been explored and used widely: tapping and lassoing. Tapping provides a fast method for selecting discrete targets such as a single stroke or symbol [5]. A study done by Mizobuchi comparing the tapping and lassoing selection techniques found that lassoing was faster than tapping for highly cohesive targets with low shape complexity. For example, lassoing a collection of strokes that form the character "H" is a lot easier than tapping each individual stroke because the strokes are all close together. However, if there were many strokes around the "H", lassoing might take longer since steering along a narrow path is slow and error prone. I decided to implement both tapping and lassoing so that users can choose the easiest method to make a selection depending on the type of selection they are trying to make.

2.4 Erase

Scribble erase gestures are a good way to indicate an erasure without switching into a special mode. Dahmen [1] developed a method for recognizing scribble erasure gestures by looking at density, speed, number of intersections, and bounding ratio. Bounding ratio was defined as the ratio of the circle width divided by the scribble width. They concluded that bounding ratio and density were the fastest and most accurate metrics to identify a scribble erase gesture. However, bounding ratio is specifically designed to distinguish between scribble erasure gestures and filling in gestures in circles. In the chemical sketch domain, the fill in gesture is used for wedge bonds, but because wedge bonds are not circular, the bounding ratio criterion does not work well. The number

of intersections was found to be decent metric though it was more time intensive to calculate than bounding ratios.

Further work in identifying scribble gestures was done by Li [3], who used entropy, least square error and number of intersections to detect scribbles. The entropy metric measured the amount of curves and angles in a potential scribble. However, text tends to have an entropy level similar to scribbles, which meant that the entropy metric was not very effective at differentiating text from scribbles. In addition, the least square error metric was only really useful for differentiating scribbles from over-traced strokes.

Chapter 3

Display Features

One important aspect of chemical diagrams is how the atoms are connected. To display that the system understands a particular atom is connected to a bond or two bond lines are connected, the system converts the bond stroke into a straight line that is connected to the appropriate bonds (Figure 3.1) and atoms (Figure 3.2). In ChemInk, there are Connection Points, which are the endpoints of bonds or the center of a symbol. These Connection Points are grouped into Connection Nodes, which indicate which Connection Points are actually connected.

(a) (b)

Figure 3.1: (a) These 2 bonds get interpreted as connected. (b) To indicate that they are connected, the system extends the endpoints until they touch

■ 3.1 Bond - Bond Connection

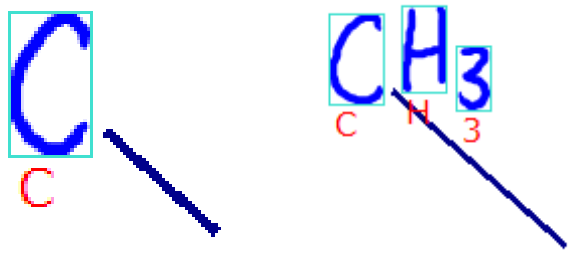
Carbon atoms are often not shown explicitly in chemical sketch diagrams. Instead, the atoms are omitted and there is an implicit carbon where the two bond lines meet, as in Figure 3.1b. Even though there is a carbon atom between the bonds, we refer to this

as a bond-bond connection.

To determine the endpoints for the bonds in a bond-bond connection, we find the midpoint between of all the Connection Points, and use that as the endpoint for each of the bonds.

■ 3.2 Bond-Symbol Connection

When a Connection Node contains symbols and bonds, it is considered a bond-symbol connection (Figure 3.2). In a bond-symbol connection, there can be more than one symbol, so we must first determine which symbol the bonds are connected to. For each bond, we determine which symbol is closest to a bond by comparing the distance from the center of each symbol in the Connection Node to the line formed by joining the endpoints of the bond. This method performed better than comparing the distance between the center of each symbol and the endpoint of the bond, because some symbols could be geometrically closer to the end of the bond, but the bond was not actually pointing to those symbols. For example, in Figure 3.2b, the end point of the bond is closer to the H, but the line formed by the bond passes closer to the C, therefore the system determines that the bond should point to the C.



(a) (b)

Figure 3.2: Two examples of a bond connected to a bond-symbol connection. In (b) the system must first determine the symbol that the bond is pointing to, in this case the C, before modifying the bond to point to the center of that symbol.

Once the closest symbol is determined, the bond is modified to point directly at the center of this symbol. However, we do not want the stroke to intersect the symbol, so we have the bond end just before the bounding box of the symbol.

■ 3.3 Double Bond

For double bonds, there are 2 bonds that both start and end at the exact same point. In order for them to not be drawn on top of each other, each bond in a double bond must be shifted to the side by a few pixels. First, we pretend the double bond is actually a single bond and determine the appropriate endpoints. Then we take these endpoints and shift them a few pixels perpendicular to the direction of the bond. In addition, we shorten the length of the bond by a few pixels so that bonds connected to the double bond do not intersect the bonds in the double bond (Figure 3.3).

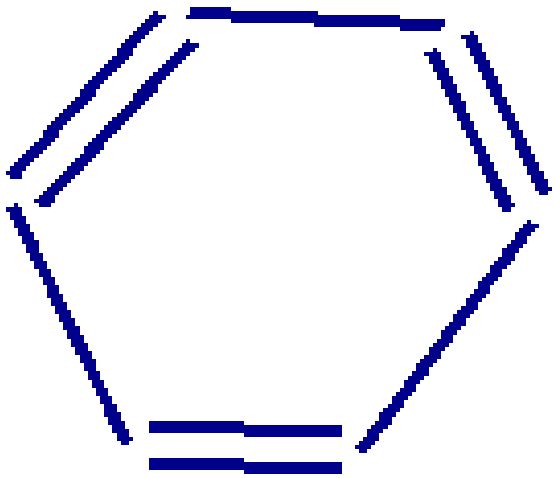


Figure 3.3: The double bonds in the benzene ring are shifted to the side so that the two bonds are not drawn on top of each other. In addition, the double bonds are shortened a bit so they don't overlap with the bonds they are connected with.

■ 3.4 Triple Bond

Similar to the double bond, triple bonds have 3 bonds that start and end at the same point. We use the same algorithm as for the double bond, except that for the third bond, we draw the bond where the original single bond would have gone. Basically, we set the endpoints of the third bond to the location as if it had been a single bond (Figure 3.4).

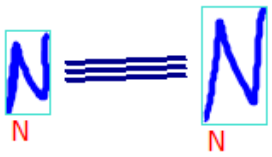


Figure 3.4: An example of a triple bond.

Chapter 4

Correction Features

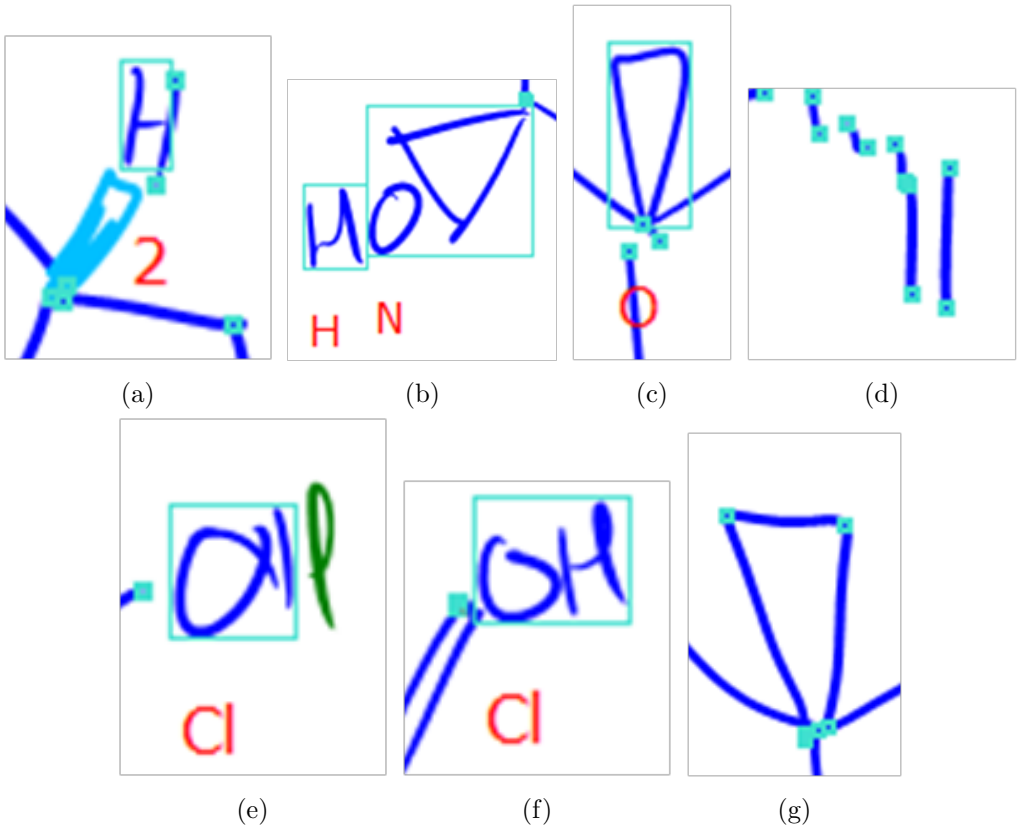
In this chapter, we discuss the various features implemented on top of an existing sketch recognizer, ChemInk, to allow the user to interact naturally with the sketch and make corrections, whether the errors are made by the user or the sketch recognizer. For example, in figure 4.1c, the sketch recognizer labeled a wedge bond as an O. In figure 4.1e, the user's messy OH is misinterpreted as a Cl.

A sketch is composed of strokes, pen movements made by the user without lifting the pen. The recognizer detects all the corners in a stroke and uses them to subdivide the stroke into segments. These segments are then clustered together by the sketch recognizer to form symbols such as chemical elements (H, Cl, N) or bonds (hash bond, wedge bond, double bond).

■ 4.1 Selections

The first task in correcting an error is to select the section of the sketch that needs to be corrected. Two traditional methods of selecting strokes in a sketch were implemented: tapping and lassoing. These two methods can both be used without switching into a special mode, meaning the user simply performs the appropriate gesture while drawing and the system understands that the user is making a selection and not trying to add more ink.

To use the tapping method of selection, the user presses the pen on the stroke he



(a) (b) (c) (d)

(e) (f) (g)

Figure 4.1: Examples of errors made by the sketch recognizer.

wishes to select. The program decides that the action is a tap if the pen travels less than 5 pixels before being lifted. Once the action is determined to be a tap, the stroke closest to the tap (within a 50 pixel radius) is selected. If that stroke is part of a symbol identified by the sketch recognizer, the entire symbol is selected.

To select strokes using lassoing, the user draws a circle around the strokes he wishes to select. There are 3 criteria used to determine if a stroke drawn by the user is actually a lassoing gesture:

1. **A Subsection of the Stroke is a Loop**: The loop must have end points that are fairly close to each other (1/5 of the diagonal length of the bounding box of the original stroke). Figure 4.2 shows examples of strokes that contain a subsection

(a) (b) (c) (d)

Figure 4.2: Examples lasso gestures. The red dots indicate the calculated end points of the loop.

that is a loop. The end points of the loop are highlighted as red dots. When a user draws a loop around an object, the 2 ends of the loop may not actually touch (Figure 4.2c). This criterion allows the loop to be recognized even if it is not fully closed. By allowing the loop to be a subsection of the stroke, the user does not need to make sure the stroke starts and ends at the same point. This also allows the loop detection to be fairly robust to pen-drag of the sort shown in Figure 4.2b, 4.2c, 4.2d.

2. **Large Loop**: The distance from the starting point to the farthest point on the loop must be at least the length of the diagonal of the original stroke. This ensures that the loop is fairly large compared to the original stroke rather than a small loop in the middle of a larger stroke containing other things, as in Figure 4.3.

Figure 4.3: An R contains a loop, but is not interpreted as a lasso because it is too small.

3. **Contains strokes**: Within the bounds of the loop, there must be other strokes, the ones being selected. These strokes do not have to be fully contained within the loop; as long as the majority by length of it is within the circle, it is considered to be contained by the circle. This allows users to casually circle strokes without worrying about including the very ends of the strokes they are trying to select into the loop.

Figure 4.4: An example of a lasso selection where the selected stroke, in this case an N, is mostly contained by the lasso. Even though the ends of the stroke are outside of the lasso, the symbol is still considered to be inside the lasso and will therefore be selected.

There are a number of different granularities at which parts of the sketch can be selected.

1. **Symbol**: Selecting symbols allows users to quickly identify mislabeled strokes that have been correctly grouped into a symbol (Figure 4.1c). Tapping allows users to select entire symbols. When a user taps on a stroke, the entire symbol that the stroke is associate with is selected.

2. **Stroke**: Selecting at the stroke level is a compromise between segments and symbols as they are the general level needed for most selections. For example, selecting at a symbol level would not work for Figure 4.1b because the sketch recognizer has grouped the O and the wedge bond into a single symbol, but selecting at the stroke level allows the user to select just the O or just the wedge bond.

Lassoing allows users to select at the stroke level. Users may use a lasso to select specific strokes within a symbol and separate them from the rest of the strokes in a symbol.

3. **Segment**: Selecting segments is necessary if the user wants to indicate that only a segment of a stroke is part of a particular symbol. Since this is not very common in normal use, a method for selecting segments was not implemented. However, in the future, we would like to make it possible for users to select segments.

■ 4.2 Correct Classification

If the sketch recognizer misinterprets a section of the sketch, the user can correct the error by manually labeling that part of the sketch. The user will select the appropriate strokes or symbols by tapping or lassoing. Then the user selects the correct labeling from a menu that appears. In the early stages, the user could access the context menu by pressing and holding the pen to simulate a right click. However, during our second user study, many users struggled to open the menu (Section 5.2.1). In the final version of our program, we moved the menu to the left side of the window (Figure 4.5) and had it appear whenever any strokes were selected. We changed the menu from being a context menu to being a side menu so that the menu would not obstruct users if they were still trying to select more strokes.

■ 4.3 Reattempt

Another method of correcting a misinterpreted section of a sketch is to tell the system that it has made a mistake and have it determine the next most likely interpretation of the sketch. To do this, the user can select any strokes in a symbol that have been misinterpreted and requires a reinterpretation. The system will find all the symbols that contain any of the selected strokes and tell the recognizer that those symbols are

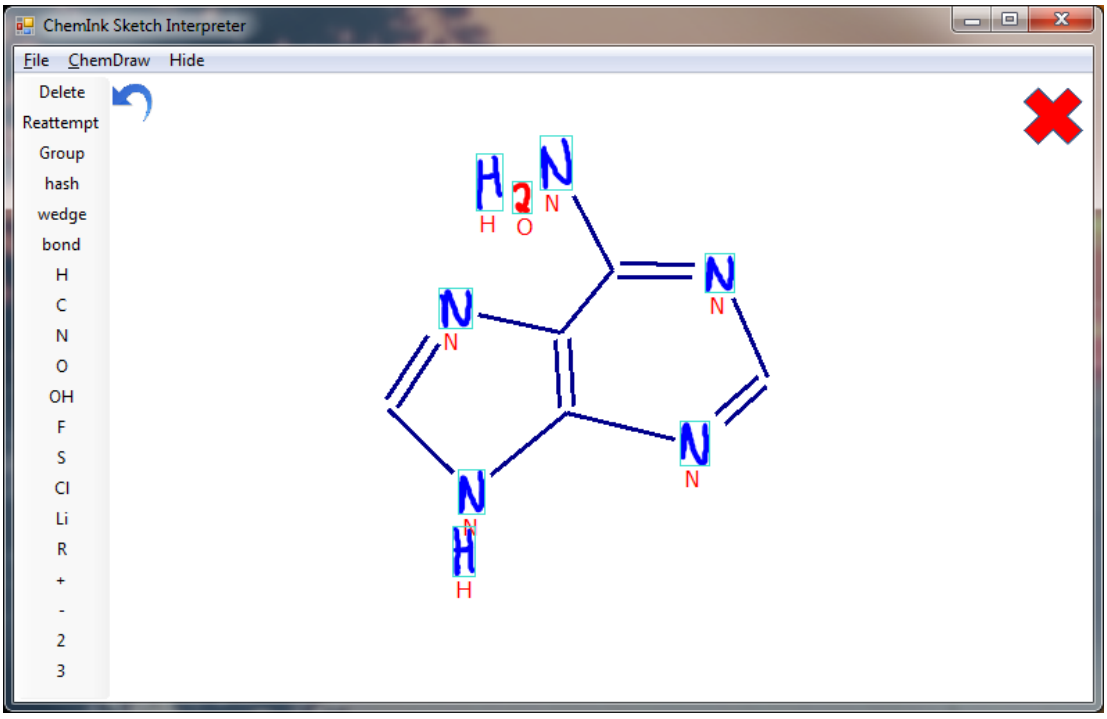


Figure 4.5: To correct the classification, the user can highlight strokes or symbols and select the correct labeling from the menu that appears on the left. In this figure, the user has selected the 2 which was incorrectly labeled as an O. The user will next select the 2 from the menu on the left.

incorrect. The sketch recognizer will take the indicated incorrect symbols out of its list of candidates and produce the next most globally likely interpretation of the sketch.

■ 4.4 Erasing

Erasing is an important part of error correction. A user might want to erase for a number of reasons. Perhaps the symbol he drew was too messy to be recognized correctly. The user might want to redraw a part of the sketch to clarify what he was trying to convey. In addition, if the user simply made a mistake while drawing, he would also want to erase a part of the sketch. Like selections, the user doesn't need to go into a special mode to erase. This allows for seamless transition between sketching and erasing.

In our implementation, we allow users to erase strokes, but not segments or pixels. Due to the way strokes are stored in the system, in order to erase part of a stroke, the system would need to delete the original stroke and redraw the portions of the stroke that were not deleted. This would add a lot of complication to the system and therefore has not been implemented yet.

Two methods for erasing were implemented. The user can use the back of the stylus to indicate the strokes he wishes to erase, or use a scribble gesture over the strokes to be erased.

To identify a scribble gesture, the algorithm must first detect the number of sharp corners in the stroke. To detect corners, the angle (θ) between each set of 3 consecutive points ($p1, p2, p3$) from the stroke is measured using Equation 4.1. If this angle is less than 30 degrees, it is considered a sharp corner. Strokes with at least 3 sharp corners are candidates to be classified as scribbles.

$$\theta = \arccos\left(\frac{a \cdot b}{|a||b|}\right) \tag{4.1}$$

$$a = (p2.x - p1.x, p2.y - p1.y)$$
$$b = (p2.x - p3.x, p2.y - p3.y)$$

We next determine which strokes the user means to erase with the scribble. When the user wants to erase a large section of the sketch, he may draw a large scribble gesture across the general area he wishes to erase, as in Figure 4.6. To determine that the user intends to erase a large section of the sketch, the diagonal length of the bounding box of the scribble must be significantly larger than that any of the strokes it is erasing. Due to the sparseness of the scribble gesture compared to the large area he is trying to erase, some of the strokes in this area might not actually touch the scribble stroke. In

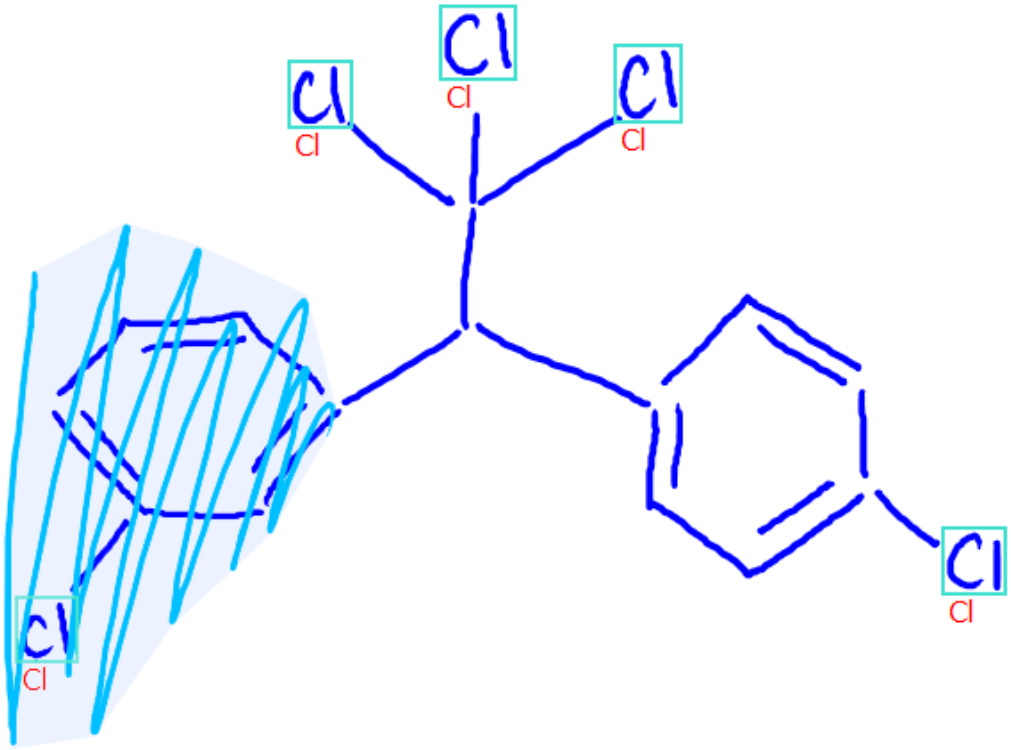
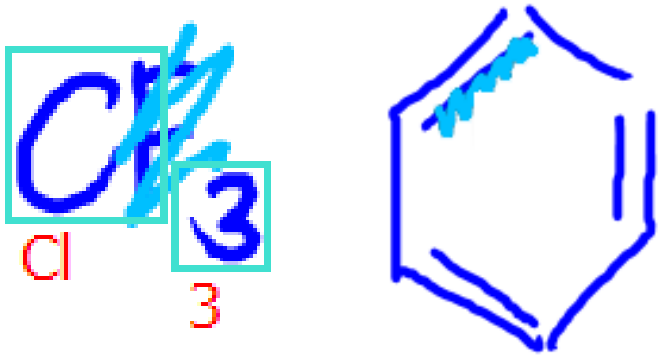


Figure 4.6: Example of a large scribble gesture in which the user intends to erase both the benzene ring and a Cl. The convex hull of the scribble is area that has been shaded light blue. All strokes for which the majority by length of the stroke is within this convex hull is erased.

order to erase all such strokes, we erase all strokes for which the majority by length of the stroke is located within the convex hull of the scribble stroke.

Other times, the user wants to carefully erase one or more specific strokes, erasing only the strokes that the scribble intersects with (Figure 4.7). During user studies (Section 5.2.1), we realized that there were 2 situations in which the system might incorrectly identify some strokes as erasure gestures, causing that stroke and any stroke it intersected to be erased.



(a) Erasing the mislabeled F (b) Erasing a bond

Figure 4.7: Examples of a erasure gesture (shown in light blue) used to erase a few specific strokes.

- If a user is trying to write the letter N (a symbol that looks very similar to the scribble) at the end of a bond (Figure 4.8), the N may accidentally intersect with the bond. The system could incorrectly classify the N as an erasure gesture and proceed to erase the bond, as well as not drawing the N. To prevent this from occurring, we added the criterion that the point at which the scribble intersects the stroke must be closer to the center of that stroke segment than to its ends.

- A false positive may also occur when a user tries to draw a wedge bond (Figure 4.9). If the user draws the outline of the wedge bond first and then tries to fill it in, his fill-scribble might touch the outline and be classified as a scribble erase gesture for the outline. To prevent this, we added the criteria that the point at which the scribble intersects the stroke must be closer to the center of that scribble segment than the ends of the scribble segment.

Essentially, to prevent false positives, the point at which the scribble and erased stroke intersect must not be too close to a corner, in either the scribble or the erased stroke. The scribble should intersect the stroke in the middle of a segment and the

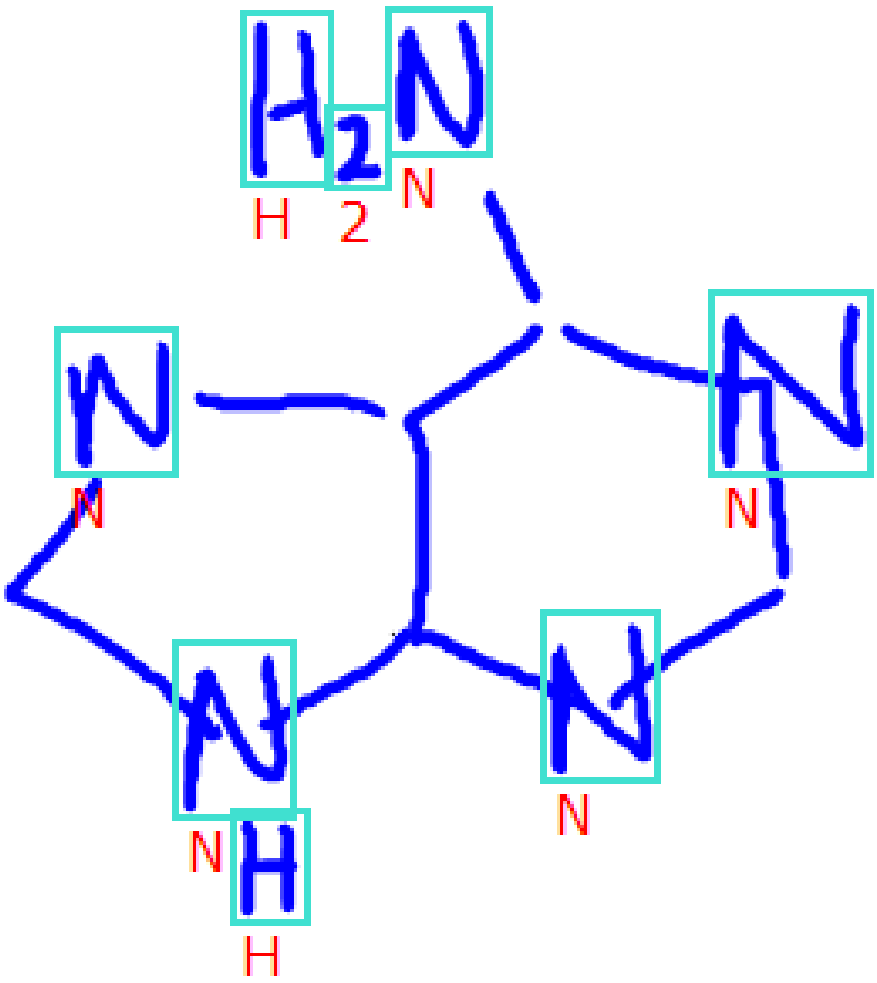


Figure 4.8: In this sketched diagram of Adenine, there are 4 occurrences when an N was drawn over a bond. These Ns could be misinterpreted as scribble erase gestures to erase the bond it intersects with if the distance from the intersection to the middle of the bond was not considered.

stroke should intersect the scribble in the middle of a segment.

If it is determined that the scribble doesn't intersect any other strokes, it is handled as a normal ink-depositing stroke.

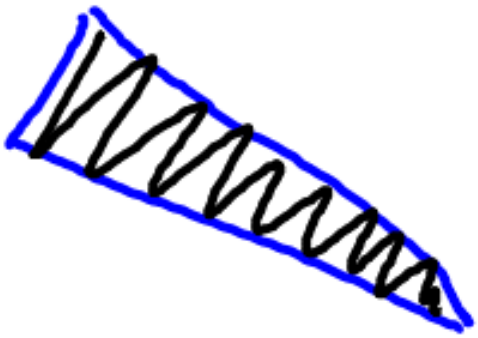


Figure 4.9: The scribble (drawn in black) in the middle of the wedge bond could be misinterpreted as an erasure gesture if the distance between the point where the scribble intersects the outer lines and the center of each segment of the scribble is not considered.

■ 4.5 Overwrite

Another method of correcting errors is to redraw the correct symbol more clearly and accurately on top of the original strokes. Since the strokes are all digital ink, when the system detects that the user is trying to overwrite a previous stroke, it will delete the original strokes.

The system detects a potential overwrite when a stroke is drawn that overlaps previous strokes or symbols. For example, in Figure 4.10b an overwrite would be detected because the new stroke overlaps the previous strokes. If a potential overwrite is not detected, the ChemInk sketch recognizer proceeds as normal to create candidate symbols for each grouping of temporally and spatially contiguous strokes. Temporal candidates are formed by sequences of consecutively drawn strokes. Spatial candidates are formed by strokes that are located close to each other. Once all the candidates are created, they are evaluated and the globally optimal set of candidates is selected that ensures that each stroke is included in exactly one candidate.

The most challenging aspect is to be able to accurately detect the overwrite action and determine which strokes were part of the original symbol and which are part of

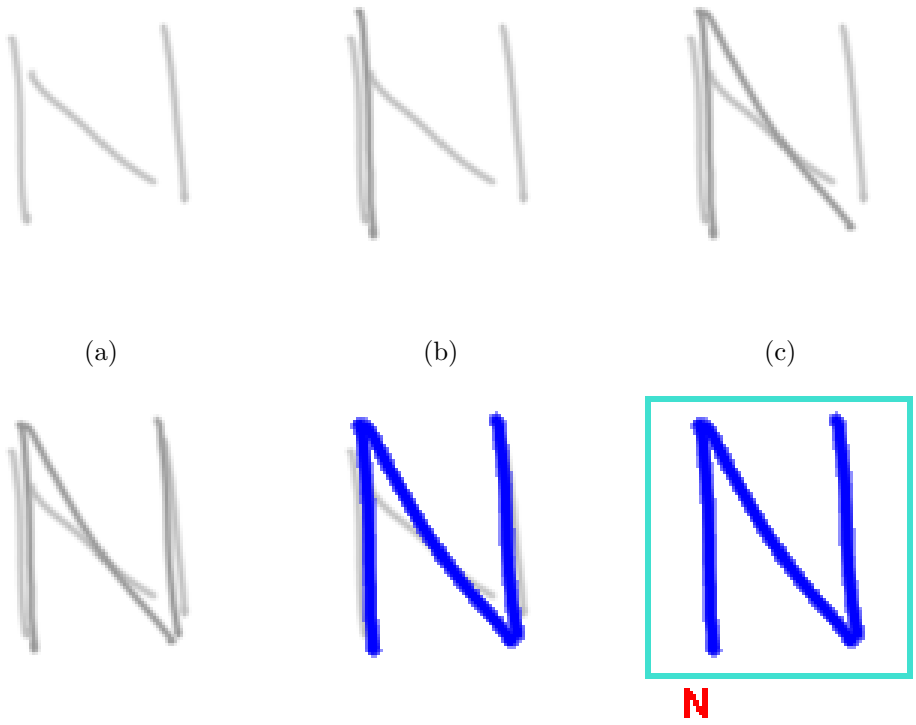


Figure 4.10: (a) The user first writes a sloppy N, but the recognition system fails to recognize it (b-d) The user then proceeds to overwrite it with a more neatly written N. (e) The system recognizes the overwriting strokes as an N. (f) Once the system recognizes an overwriting symbol, it deletes the overwritten strokes.

the new symbol. If a potential overwrite is detected, the system begins evaluating possible overwrite candidates. Overwrite candidates are essentially candidate symbols that contain a special list of "new overwriting strokes", which are the only strokes looked at when trying to predict the label of that candidate. We take a group of strokes that are drawn near each other and play the strokes in reverse chronological order to see if it forms a symbol (Figure 4.11). Then, we give all these overwrite candidates to the sketch recognizer and the most probable candidate is chosen. We harness the power of the sketch recognizer to determine the most probable symbol that the user meant to

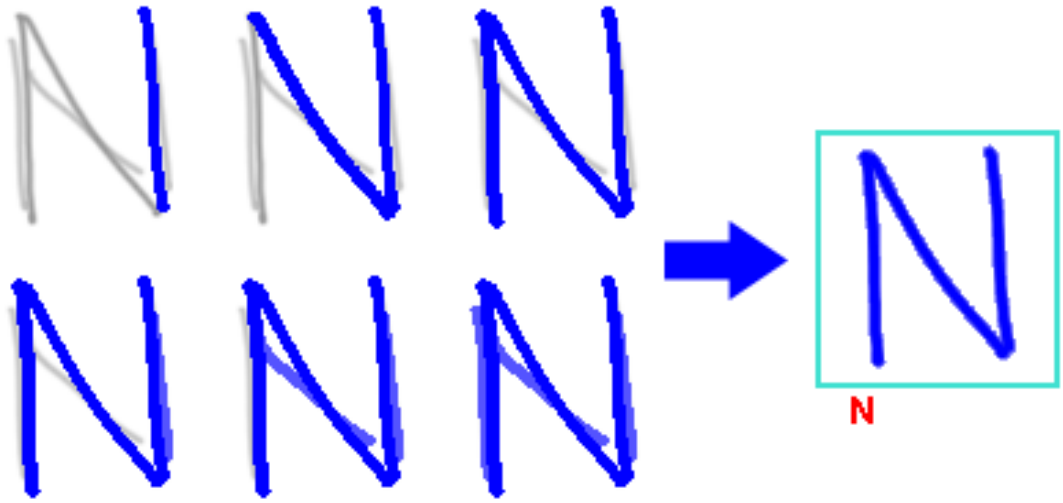


Figure 4.11: On the left are the overwrite candidates created for the example of overwriting an N. In each candidate, the new overwriting strokes are indicated in blue, while the original strokes are in grey. Once the recognition chooses the most probable overwrite candidate, the original strokes are deleted and the new overwriting strokes are labeled (shown on the right).

draw.

Once the most probable candidate is chosen, the system displays the new overwrite strokes and deletes the original strokes (Figure 4.10f).

■ 4.6 Undo

There are cases in which the user may make a mistake or that the system does something the user did not expect. For example, the system may believe the user is trying to erase something through a scribble gesture when in fact he is trying to write an N. In these cases, the user can press the undo button to have the last action undone. Undo and redo are both common features found in most types of editing software. We implemented a single level undo, which allows users to undo only one previous action.

4.7 Zoom

If the user wishes to make very fine-grained changes or add something into a tight location, he can zoom into that area by pressing the zoom in button and pressing the location he wants to work on (Figure 4.12). Once zoomed in, the user can continue to manipulate the sketch as before, erasing or adding new strokes with more fine grained control. This allows the users to squeeze in things that might not fit otherwise.

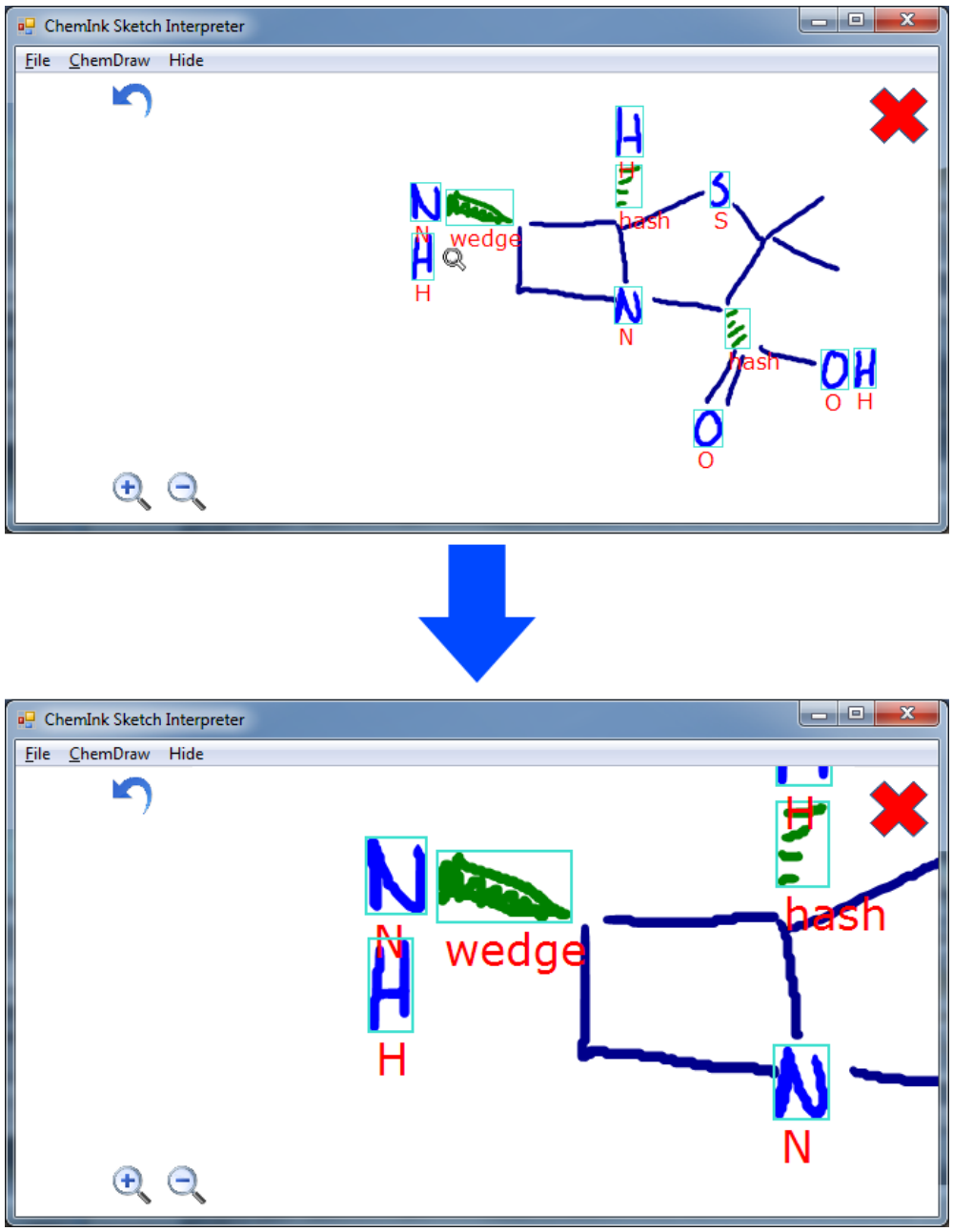


Figure 4.12: In this example, the user tapped the location near the NH, which caused the display to zoom in on that location.

Chapter 5

User Study

We conducted 3 user studies to select and evaluate the sketch editing features for our sketch recognizer. These studies were used to both determine the natural way in which users wish to interact with a sketch recognizer, and study how easy to learn and intuitive our implemented sketch editing features are.

The purpose of the first study was to determine what editing capabilities were needed, while the second and third studies were used to determine the most intuitive sketch editing techniques, as well as test the implemented editing features.

5.1 User Study 1

In the first study, we gave a class of 30 introductory Chemistry students tablets during one of their recitations. With the instruction of the TA, the students proceeded to draw in ChemInk the chemical diagrams they were learning. As the students drew, ChemInk interpreted and labeled the strokes. Once the tablet had successfully recognized the sketch, the students could export the diagram to ChemDraw, where they could view the chemical in 3D.

5.1.1 Discussion

Overall, the students enjoyed working with the tablets and liked how easy it was for their hand drawn sketches to be converted into ChemDraw CAD models of their molecules.

Erasing

However, the recognizer did make many mistakes and students would often have to correct it by erasing and redrawing that region. Students used the back of the stylus to erase and found that to be a bit slow and inaccurate. Sometimes portions of the sketch that were near the error would get erased by accident, while other times the student had to try multiple times to erase a particular stroke. When users tried to erase large sections of strokes, some of the smaller strokes were missed and left behind. The user then had to go through and individually erase each of these strokes.

These observations led us to develop two alternate ways to erase strokes. The first method is to allow users to select strokes by clicking or lassoing, and then selecting the delete button from a menu. This method allows users to first see which strokes they will be deleting and minimizes the possibility of the user accidently erasing strokes. The second method is the scribble erase gesture, which allows for more fluid and faster erasures since users no longer need to flip over their pen in order to erase. This method also allows users to erase large sections of strokes in a single gesture.

Correcting Labeling

The recognizer would at times continue to have trouble recognizing a particular symbol, even after the user erased and redrew it. This may be because the user drew that particular symbol differently from any of the training examples that the recognizer had seen before or it may be that the user was particularly messy. The result was that the user often became frustrated as he continuously erased and redrew a symbol in hopes that the recognizer would accurately recognize it this time. This was particularly frustrating to the user because the system did not provide any feedback as to why the symbol was not properly recognized or how the user could change his behavior to help the recognizer understand the symbol better.

Figure 5.1: A user participating in our user study by drawing chemical molecule diagrams in ChemInk and testing its editing capabilities.

In order to resolve this problem, we developed a number of ways for the user to give feedback to the recognizer including manually labelling symbols, telling the system to reattempt or overwriting the error.

■ 5.2 User Study 2

In the second study, 7 Chemistry student volunteers were asked to use a Windows tablet running the ChemInk program. Through a 3 phase study, we determined how users intuitively wanted to edit sketches, as well as how natural and easy to learn some of

our implimented editing methods were. These editing methods included lasso selection, click selection, labeling using a context menu, reattempt, and an early implimentation of scribble to erase.

During the first phase, the user was given the diagrams of 4 chemical molecules (Appendix A) and asked to draw each molecule on the tablet, with no other instructions. When an error was noticed by the user, the facilitator would point it out and ask the user what he/she would like to do in order to correct it. The goal of this was to determine how the user would intuitively like to make corrections to sketch errors.

In the second phase, the facilitator explains and demonstrates to the user each of the editing technique that was implemented in ChemInk. The user is then given the diagrams of 10 chemical molecules (Appendix A) and asked to draw them, using any of the demonstrated editing techniques in order to correct any mistakes made by either the recognizer or by the user. This allows users to learn and test out each of the editing techniques and determine how natural it feels to use each of them.

Finally, in the third phase, the user is given a standard set of pre-drawn chemical diagram sketches that contain recognition errors. The user is asked to correct each of these errors. This phase of the study ensures that all users will correct the same errors, in order to allow us to compare how different users tackled correcting the same error.

Following the completion of these 3 phases, each user was asked to fill out a questionnaire. In the questionnaire, users were asked to rate on a likert scale of 1 to 5 how natural and intuitive each of the implemented correction methods felt, where 1 meant unintuitive and 5 meant intuitive.

5.2.1 Experimental Results

Intuition

In the first phase of the user study over half the users (4 out of 7 users) suggested overwriting as a means to correcting errors. This overwhelming intuition for overwriting errors motivated us to implement overwrite before our 3rd user study. Three of the users intuitively wanted edit by erasing and redrawing erroneous parts of the sketch. One user suggested tapping on the error and relabeling it. (This does not add up to 7 because one of the users suggested multiple correction methods.)

Context Menu

The biggest problem that many of the users faced was difficulty in accessing the context menu (Figure 5.2). Most of the functionality of the system, including labeling and reattempt were accessed through a context menu. This context menu would appear when the user selected some strokes and proceeded to click and hold the pen, which is how a pen simulates a right click. Unfortunately, detecting the right click, which is done by the Windows 7 operating system, depended on careful timing of the length of time the user held down the pen, as well as lifting the pen straight up and not moving laterally. Because of how finicky the system was in detecting a right click, many users struggled to get the context menu to open. What should have been a trivial task of opening a menu, seemed to become an arduous challenge, as users tried again and again to get the context menu to appear.

Scribble Erase

The scribble erase feature caused many problems for users. As described in Section 4.4, the system would often identify the letter N drawn at the end of a bond as a scribble and erase both the N and the bond. Other times, if a user filled in a wedge bond, the

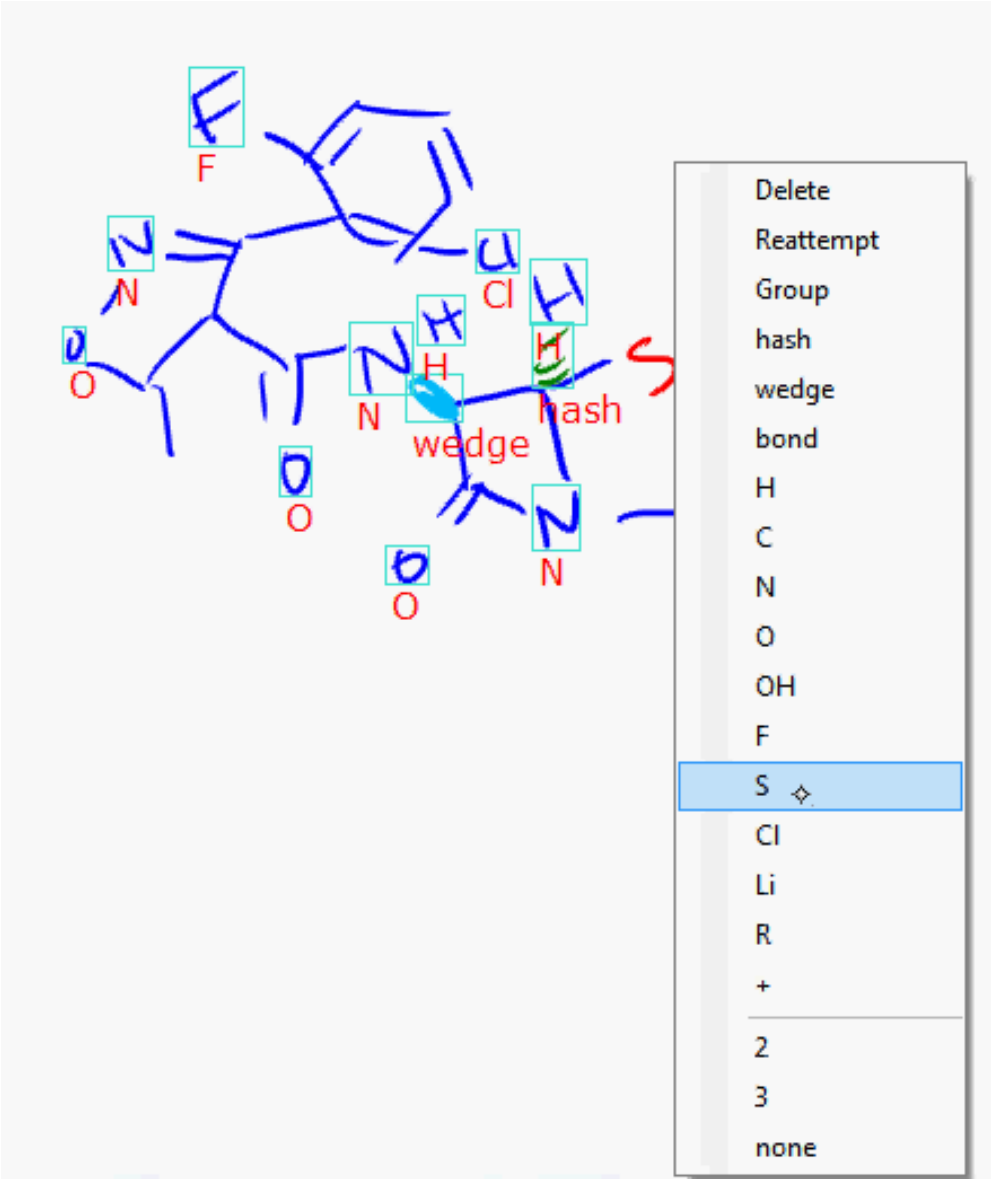


Figure 5.2: The context menu was extremely difficult to open due to the challenges of detecting a right click.

system would misinterpret these actions as a scribble and also delete the entire wedge bond.

Connectedness

Many of the users suggested that the system should indicate in some way which bonds were connected. Often times, due to a messy drawing, it was unclear if the system understood that 2 bonds were in fact connected, or if a bond was connected to a specific symbol. One user also suggested indicating that the system properly recognized double bonds.

Overall

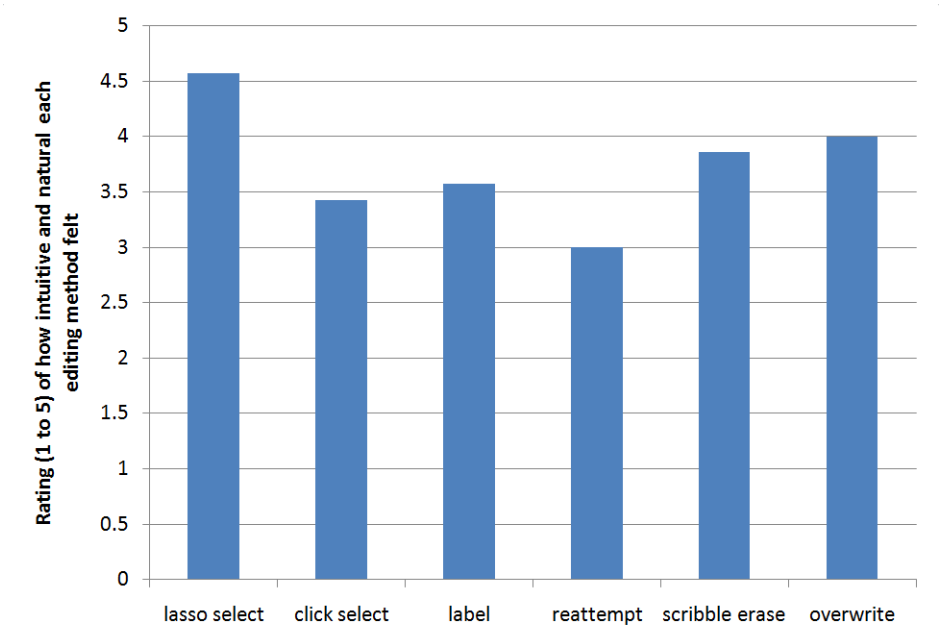


Figure 5.3: Users found the lasso selection to be the easier and most natural correction method.

In general, users found that lasso selecting and labeling to be the fastest and most natural way to correct errors in this version of the sketch editor (Figure 5.3).

5.3 Study 3

The third user study was very similar to the second user study, but was run on a version of ChemInk that implemented more editing features. There were again 7 Chemistry student volunteers who participated in the 3 phase study. The 3 phases were exactly as described in Section 5.2. The main differences were the features that were implemented in the system, including overwrite, a more robust scribble erase algorithm, a side menu instead of a context menu and a new visualization for bonds that are connected (Chapter 3).

In addition, we added an extra section to the questionnaire at the end in which we asked users to rate how easy to learn each editing method on a likert scale of 1 to 5.

5.3.1 Experimental Results

Intuition

During the first phase of the third user study, users were asked how they would like to correct a recognition error. Three users suggested overwriting the error, 4 users wanted to erase and redraw the error and 1 user tried tapping on the error and relabeling it. (One of the users suggested multiple correction methods). Of the 4 people who wanted to erase and redraw the portion of the sketch with the error, 3 of them used the back of the pen to erase the error, while 1 of them scribbled out the error.

Lasso Selection

The results of the questionnaire (Figure 5.4) show that the lasso selection was the easiest to learn, as well as most intuitive editing method. By the end of most user studies, the preferred editing technique for most users was the lasso selection and manually relabeling the symbol. This contradicts how most people wanted to interact with the sketch editor, as only 2 out of 14 people had initially indicated that as the way they

would want to correct errors. I believe that the reason most people ended up liking and using the lasso selection was that it was the most consistent of all the editing techniques. Unlike the other techniques that successfully correct the error only some of the time, lassoing and labeling always produces a correctly labeled symbol.

Click Selection

Users felt that click selection was not as natural as lasso selection even though they both allow the user to directly select which strokes to classify. However, click selection was less favored because the user had to be more precise by tapping directly on a specific stroke. In addition, when selecting multiple strokes, the lasso is a lot faster.

Labeling

Like the click selection, users did not like the act of manually labeling a symbol due to the inaccuracy of the tip of the pen. Users often had trouble clicking on the correct label on the menu. In addition, searching through the list of all the possible labels was time consuming and unintuitive.

Reattempt

Reattempt was easy to learn, but users felt it was unintuitive. Some users wanted to be able to press reattempt over and over again instead of needing to reselect the strokes. Other users felt it was frustrating when the system would not get the correct labeling after trying reattempt several times.

Scribble Erase

The scribble erase was slightly more difficult to learn to use as the scribbles needed to have sharp corners in order to be detected. However, once learned, users felt that it was very natural and intuitive. Once they learned about the scribble erase, they often

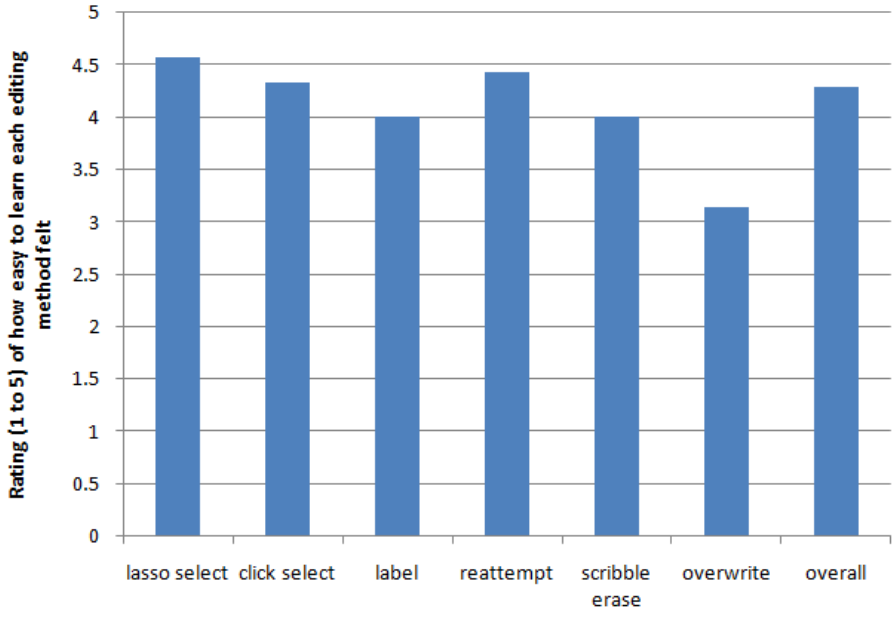
preferred it over erasing using the back of the pen.

Overwrite

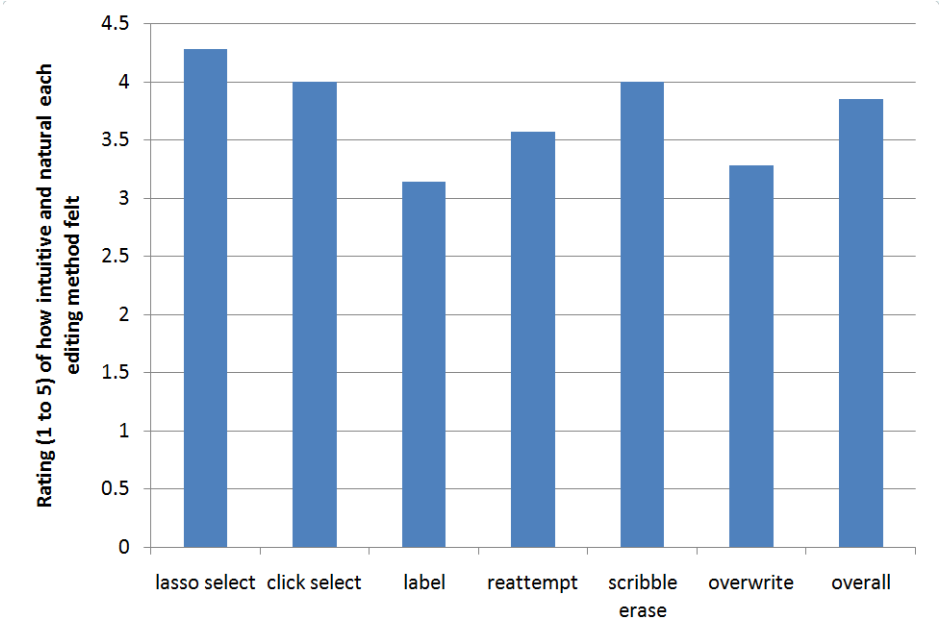
Even though the overwrite method tied with erasing and redrawing as the most commonly suggested editing method, users found it to be most difficult to learn and least natural to use. This is probably because the current implementation of overwrite is not very robust and has a fairly high failure rate. Most users tried it one time and when it did not immediately work, they never tried overwriting again.

Connectedness

While users liked the idea of showing which bonds are connected, they found it frustrating that bonds that they had just drawn would move before they were able to finish drawing the structure. Users were also frustrated by the number of incorrect connections displayed. These inaccurate interpretations were also present in the older versions of the software, but were not displayed to the user. The fact that we now display what bonds are connected allows users to visually see these errors and try to correct them. However, we do not yet have a good mechanism for correcting bond connectedness.



(a)



(b)

Figure 5.4: Average rating of how easy to learn (a) and how natural and intuitive (b) each editing method felt

Chapter 6

Conclusion

6.1 Lessons Learned

The work in this thesis demonstrated and tested various ways to allow users to interact with a sketch recognition system.

We learned that selecting and manually classifying symbols was not the first choice for error correction, but it turned out to be the most used editing technique because it was robust and consistent. The algorithm we developed to detect lasso selections allowed the user to effortlessly circle some strokes without worrying about getting the ends of the stroke into the circle, or perfectly closing the circle.

Erasing and redrawing a symbol was a natural and intuitive correction method, but had less consistent results, which frustrated some users. The algorithm we developed for detecting scribble erase gestures was fairly intuitive, but would occasionally not detect the scribble. It is also not guaranteed that the system will correctly recognize a symbol after the user redraws it, causing some users to be stuck in a loop of erasing and redrawing a symbol over and over again.

The user studies showed that overwriting errors is an intuitive way that users want to interact with a sketch editor. However, due to the current state of technology, the system does not consistently recognize overwrites. Due to the messy nature of overwriting an error, users tend to not retry the overwrite technique after it failed the

first time. Therefore, in order to make overwrite a successful editing technique, it needs to be very robust and accurate.

6.2 Future Work

For our future work, we would like to allow users to not only select strokes and symbols, but also segments of a stroke. This is useful for cases when a user might draw multiple bonds in one stroke, but then wishes to erase just one of the bonds. He should also be able to erase a segment through either using the back of the pen or the scribble gesture.

Currently, in order to use reattempt, the user must select each stroke they wish to be re-evaluated and hope that the next interpretation is correct. To make reattempt easier to use, we plan to create an interface that allow users to scroll through possible interpretations of a set of strokes in order to find the correct interpretation. These interpretations could involve different stroke groupings and multiple symbols. This allows users to not commit to using reattempt unless he knows it will succeed.

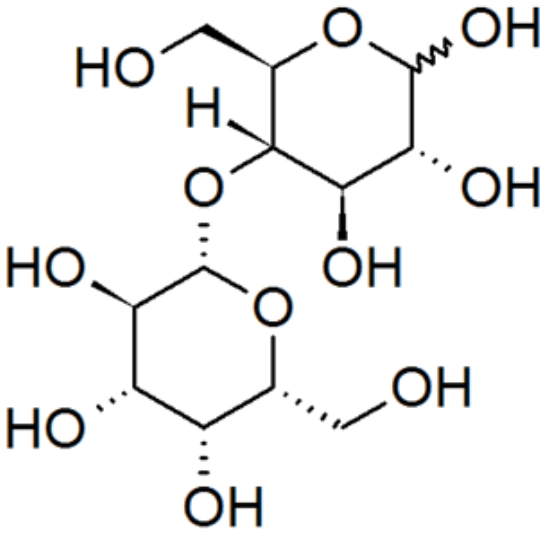
Users in all 3 user studies suggested that it would be nice to have a live ChemDraw window that displayed the computer generated diagram of the current interpretation of the sketch. This would be really good feedback to the users and help them ensure that the system was in fact correctly understanding their sketch.

We would also like to implement a method to allow users to correct errors in how the bonds are connected. We could do this by allowing users to drag the ends of bonds to where they are supposed to be connected. We could also allow users to extend bonds after they are drawn to clarify what they are connected to.

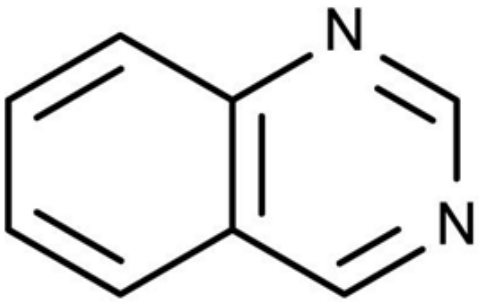
Appendix A

Chemical Molecular Diagrams

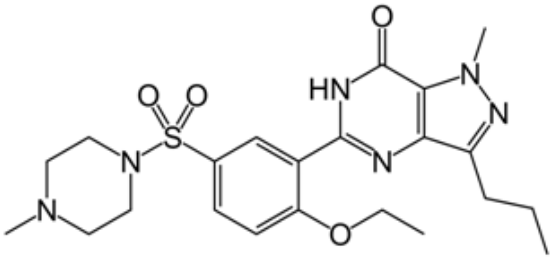
Lactose



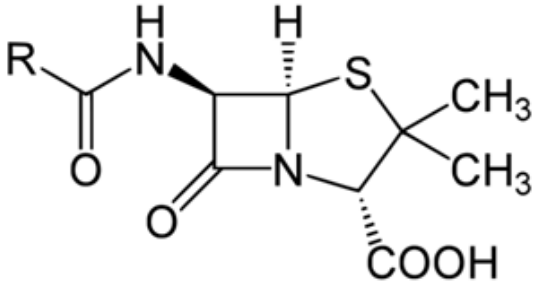
Quinazoline



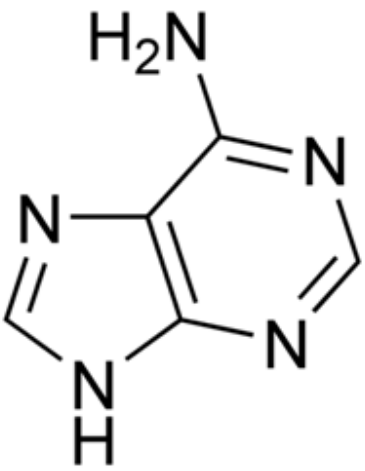
Sildenafil



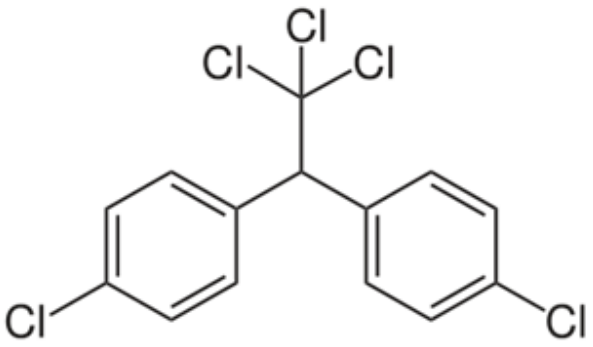
Penicillin



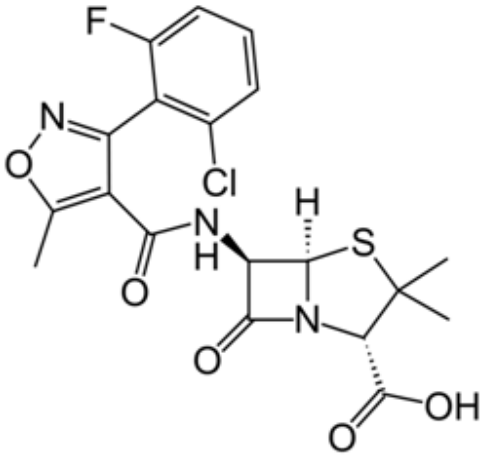
Adenine



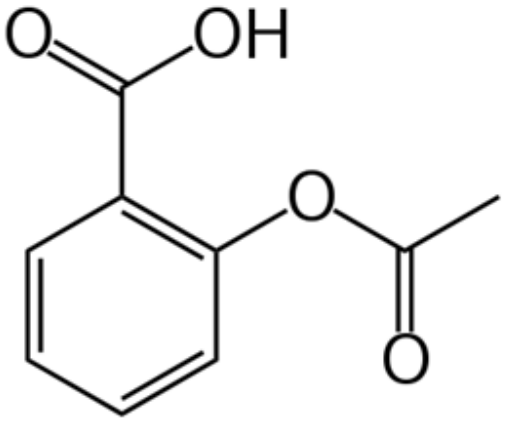
DDT



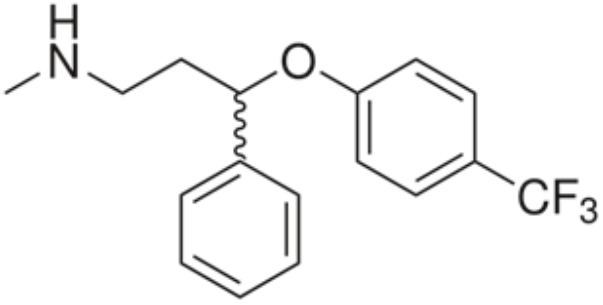
Flucloxacillin



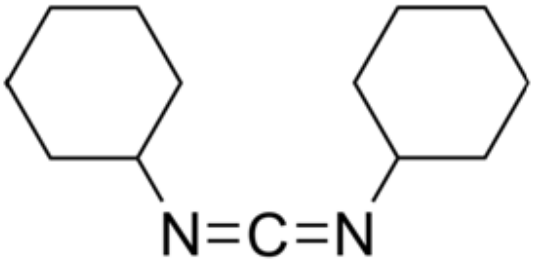
Aspirin



Fluoxetine



DCC



Bibliography

[1] Katie Dahmen and Tracy Hammond. Distinguishing between sketched scribble look alikes. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 3*, AAAI'08, pages 1790–1791. AAAI Press, 2008. ISBN 978-1-57735-368-3. URL http://dl.acm.org/citation.cfm?id=1620270.1620355.

[2] Tovi Grossman, Patrick Baudisch, and Ken Hinckley. Handle flags: efficient and flexible selections for inking applications. In *Proceedings of Graphics Interface 2009*, GI '09, pages 167–174, Toronto, Ont., Canada, Canada, 2009. Canadian Information Processing Society. ISBN 978-1-56881-470-4. URL http://dl.acm.org/citation.cfm?id=1555880.1555918.

[3] Wenzhe Li and Tracy Hammond. Using scribble gestures to enhance editing behaviors of sketch recognition systems. In *Proceedings of the 2012 ACM annual conference extended abstracts on Human Factors in Computing Systems Extended Abstracts*, CHI EA '12, pages 2213–2218, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1016-1. doi: 10.1145/2223656.2223778. URL http://doi.acm.org/10.1145/2223656.2223778.

[4] Jennifer Mankoff, Scott E. Hudson, and Gregory D. Abowd. Interaction techniques for ambiguity resolution in recognition-based interfaces. In *Proceedings of the 13th annual ACM symposium on User interface software and technology*, UIST '00, pages

11–20, New York, NY, USA, 2000. ACM. ISBN 1-58113-212-3. doi: 10.1145/354401.354407. URL http://doi.acm.org/10.1145/354401.354407.

[5] Sachi Mizobuchi and Michiaki Yasumura. Tapping vs. circling selections on pen-based devices: evidence for different performance-shaping factors. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 607–614, New York, NY, USA, 2004. ACM. ISBN 1-58113-702-8. doi: 10.1145/985692.985769. URL http://doi.acm.org/10.1145/985692.985769.

[6] Tom Y. Ouyang and Randall Davis. Chemink: a natural real-time recognition system for chemical drawings. In *Proceedings of the 16th international conference on Intelligent user interfaces*, IUI '11, pages 267–276, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0419-1. doi: 10.1145/1943403.1943444. URL http://doi.acm.org/10.1145/1943403.1943444.

[7] Gonzalo A. Ramos and Ravin Balakrishnan. Pressure marks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 1375–1384, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-593-9. doi: 10.1145/1240624.1240834. URL http://doi.acm.org/10.1145/1240624.1240834.