ChemInk: A Natural Real-Time Recognition System for Chemical Drawings

Tom Y. Ouyang
MIT CSAIL
32 Vassar Street
Cambridge, MA 02139
ouyang@csail.mit.edu

Randall Davis
MIT CSAIL
32 Vassar Street
Cambridge, MA 02139
davis@csail.mit.edu

ABSTRACT

We describe a new sketch recognition framework for chemical structure drawings that combines multiple levels of visual features using a jointly trained conditional random field. This joint model of appearance at different levels of detail makes our framework less sensitive to noise and drawing variations, improving accuracy and robustness. In addition, we present a novel learning-based approach to corner detection that achieves nearly perfect accuracy in our domain. The result is a recognizer that is better able to handle the wide range of drawing styles found in messy freehand sketches. Our system handles both graphics and text, producing a complete molecular structure as output. It works in real time, providing visual feedback about the recognition progress. On a dataset of chemical drawings our system achieved an accuracy rate of 97.4%, an improvement over the best reported results in literature. A preliminary user study also showed that participants were on average over twice as fast using our sketch-based system compared to ChemDraw, a popular CAD-based tool for authoring chemical diagrams. This was the case even though most of the users had years of experience using ChemDraw and little or no experience using Tablet PCs.

Author Keywords
Sketch recognition, chemical diagram recognition, graphical models, pen-based interfaces.

ACM Classification Keywords
H.5.2 Information interfaces and presentation (e.g., HCI): User Interfaces.

General Terms
Algorithms, Design, Human Factors, Performance.

INTRODUCTION

Sketches and diagrams are an essential means of communicating information and structure in many different domains, and can be an important part of the early design process, where they help people explore rough ideas and solutions in an informal environment. Despite the ubiquity of sketches, there is still a large gap between how people naturally interact with diagrams and how computers understand them today.

One field where sketches and diagrams are especially widely used is in chemistry. When chemists need to describe the structure of a compound to a colleague, they typically do so by drawing a diagram (e.g., the grey sketch in Figure 1). When they need to convey the same structure to a computer, however, they must re-create the diagram using programs like ChemDraw that still rely on a traditional point-click-and-drag style of interaction. While such programs offer many useful features and are very popular with chemists, these CAD-based systems simply do not provide the ease of use or speed of simply drawing on paper.

Our goal is to develop an intelligent sketch understanding system that provides a more natural way to specify chemical structures to a computer. To preserve the familiar experience of drawing on paper, our interface allows users to use the same set of standard chemical notations and symbols they used before. However, unlike real pen and paper, sketches created using digital ink are interpreted and understood by our system, which converts them to a format that can be readily exported to other tasks such as structure analysis, visualization, and database/literature search.

This paper presents a new sketch recognition framework and applies it to hand-drawn chemical diagrams. The framework combines a hierarchy of visual features into a joint model using a discriminatively trained conditional random field. This joint model of appearance makes our framework less sensitive to noise and drawing variations, improving accuracy and robustness. The key research contributions of this paper are:

- A symbol recognition architecture that combines vision-based features at multiple levels of detail.
- A discriminatively trained graphical model that unifies the predictions at each level and captures the relationships between symbols.

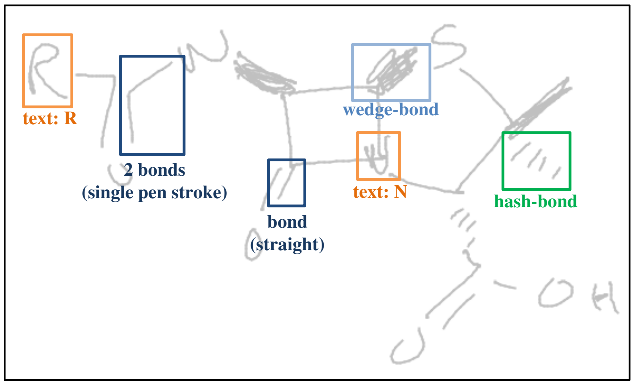

Figure 1. Grey strokes: an example of a chemical drawing that our system is designed to recognize. The notation consists of element abbreviations (e.g., "N", "O"), group abbreviations (e.g., "R"), straight bonds, hash bonds, and wedge bonds. Wedge and hash bonds show the 3-D structure of a molecule: hash bonds angle down beneath the plane, wedge bonds angle up.

- A new approach to corner detection that learns a domain-specific model of how to segment strokes.
- A new clustering-based algorithm for inferring the connectivity structure of sketched symbols.
- A real-time sketch recognition interface that has been evaluated by intended end-users and compared against the most popular existing technique for chemical diagram authoring, demonstrating a two-fold speed advantage.

HIERARCHICAL SKETCH RECOGNITION

Our system interprets each sketch using three levels of classification : inkpoints, segments, and candidate symbols.

InkPoints

The first level of the hierarchy is composed of inkpoints, data points sampled at a regular spatial interval on each stroke (Figure 2). At each inkpoint, we model the surrounding patch of ink using a set of rich local descriptors similar to those in computer vision [3, 10]. These descriptors focus on visual appearance rather than temporal or geometric patterns, making them less sensitive to stroke level differences like pen-drag (not lifting the pen between typically separate strokes) and over-tracing (drawing over a previously drawn region or shading). This improves robustness and accuracy.

Our method uses four sets of feature images to describe the local appearance around each inkpoint at varying scales and orientations. The individual feature images in each set act as orientation filters, capturing only the ink that was drawn at a specified pen direction (at 0, 45, 90, and 135 degrees). For example, in the 0-degree feature image a bright pixel indicates that the pen direction at that point is perfectly horizontal, a dim pixel indicates that the direction is somewhat horizontal, and a black pixel means that there is no ink at that point or that the pen direction is diagonal or vertical.

We make these descriptors invariant to scale by normalizing the size of the ink patch based on L, an estimate of the scale

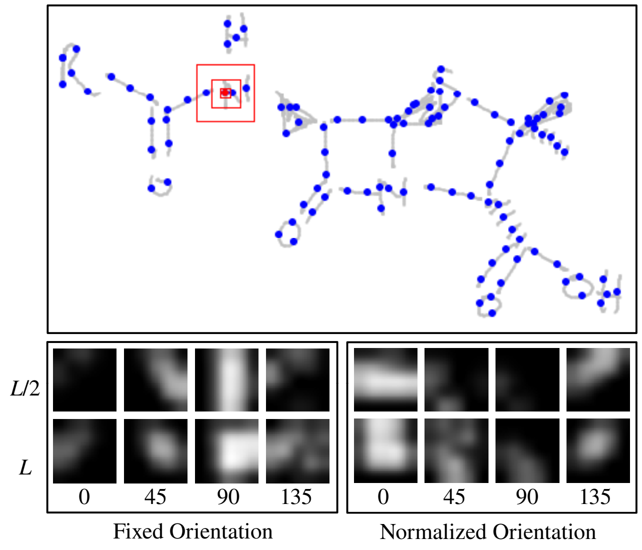

Figure 2. Shows the set of inkpoints (blue) that were extracted from a chemical diagram. For the inkpoint highlighted in red, the two red boxes show the size of the $L/2$ and L feature regions. The figures below show the set of feature images generated for the same inkpoint.

of the sketch (described in the next section). We also make half of the images invariant to rotation by reorienting them so that the direction of the pen at the inkpoint is horizontal. This dual representation helps the system model both variable-orientation symbols like bonds as well as fixed-orientation symbols like elements and group abbreviations.

The set of visual ink features are rendered onto four 10x10 pixel feature images. We perform Gaussian smoothing on each image to improve robustness and reduce sensitivity to small distortions and noise. We then downsample each image by a factor of 2 to a final size of 5x5 pixels to improve computation speed. The result is a set of sixteen 5x5 pixel images, producing a total of 400 feature values per inkpoint.

Segment Extraction

The second level of the hierarchy is composed of stroke segments extracted from the sketch. These segments are generated by dividing strokes at corner points (Figure 3). In our domain corners have a special meaning because they determine the breaks between straight bonds. This is because chemists often draw multiple straight bonds using a single polyline stroke (Figure 1), relying on the reader to infer that they are actually drawing multiple individual bonds connected by implicit Carbons.[1]

Corner detection is a well-studied problem in sketch recognition. Previous approaches have explored looking at extremes in curvature and pen speed [18], temporal patterns in pen direction [17], and alternative approximations to stroke curvature [25]. These methods often use hand-coded thresholds to achieve good performance and custom heuristics to

[1]Carbons and Hydrogen atoms are so common in chemistry that they are typically left out of the drawing, and are assumed to be present anywhere that two bonds connect without an intermediate atom.

deal with common errors. Prior work has also focused primarily on finding well defined corners in isolated shapes, where there is a clear distinction between corners, curves, and lines. However, as seen in Figure 3, corners in real-world chemical drawings are often messy and unclear.

To deal with these challenges, we designed a novel corner detection algorithm that *learns* how to segment a stroke. Instead of forcing the developer to define thresholds and parameters beforehand, we train our corner detector from labeled sketch data. This allows our detector to learn a specific model of what it means to be a corner for chemical diagrams, which may be different from what it means to be a corner in another domain. To the best of our knowledge this is the first trainable corner detector used as part of a complete sketch recognition system.

Instead of immediately trying to decide which points are corners, our system instead repeatedly removes the point that is *least likely* to be a corner. This process stops when the system decides that all of the remaining points are likely to be corners. Specifically, our algorithm repeatedly discards the point p_i that introduces the smallest cost when removed:

$$cost(p_i) = \sqrt{\text{mse}(\boldsymbol{s}_i; p_{i-1}, p_{i+1})} \cdot \text{dist}(p_i; p_{i-1}, p_{i+1}) \quad (1)$$

where $\boldsymbol{s}_i$ is the subset of points in the original stroke between point p_{i-1} and point p_{i+1} and $\text{mse}(\boldsymbol{s}_i; p_{i-1}, p_{i+1})$ is the mean squared error between the set $\boldsymbol{s}_i$ and the line segment formed by (p_{i-1}, p_{i+1}). The term $\text{dist}(p_i; p_{i-1}, p_{i+1})$ is the minimum distance between p_i and the line segment formed by (p_{i-1}, p_{i+1}).

Instead of using a hard threshold to determine when to stop removing vertices, our system learns the likelihood of a vertex being a corner from training data. For each vertex elimination candidate $\hat{p}$ (the point with the lowest cost) it extracts the set of features shown in Table 1. During classification, if the classifier decides that $\hat{p}$ is not a corner, it removes the vertex and continues to the next elimination candidate. If, on the other hand, it decides that it is a corner, the process stops and all remaining vertices are returned as corners.

One important feature of our approach is that in each iteration the system makes its decision based on the set of corner candidates that are still remaining, taking advantage of the partial solution generated so far. To illustrate this, consider the bottom ring in the left diagram of Figure 3, where there are two high-curvature points close to each other and only one of them is an intended corner (the other has high curvature due to noise, a common problem in corner detection since noise is easily mistaken for a corner). When both high-curvature points still remain in the polyline approximation, removing either one of them will not change the local shape by very much (i.e., have low cost). However, after one of them is removed, the cost of removing the remaining point becomes much larger. This leads to the correct behavior of eliminating only one of the points. Of course in our implementation the other features from Table 1 will factor into the decision, so this is an illustrative but much simplified description.

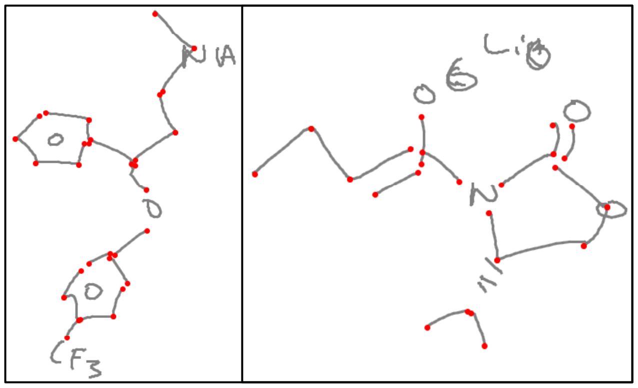

Figure 3. The result of our segment extraction algorithm on two chemical drawings. Detected corners are highlighted in red. Note that we only show corners from strokes that represent straight bonds.

Feature	Description
Cost	The cost of removing the vertex, from Equation 1.
Diagonal	The diagonal length of the stroke's bounding box.
Ink Density	The length of the stroke divided by the diagonal length.
Max Distance	The distance to the farther of its two neighbor (p_{i-1} or p_{i+1}) normalized by the distance between the two neighbors.
Min Distance	The distance to the nearer of its two neighbor normalized by the distance between the two.
Sum Distance	The sum of the distances to the two neighbors normalized by the distance between the two.

Table 1. List of features for corner detection.

After segment extraction the system records the length of the longest segment L (excluding the top 5% as outliers). This value is later used as an estimate for the scale of the sketch.

Segment Features

We compute two types of features for segments. The first type consists of the same feature images that we use for ink-points, except in this case the image regions are centered at the midpoint of the segment, and the width and height are set to L for the first scale and $2L$ for the second scale. The number of pixels in the feature images is the same as before, once again producing 400 feature values. The second type consists of a set of geometric properties listed in Table 2.

Symbols

Symbols are the final unit of classification in our hierarchy. We define a symbol as a group of one or more segments that represents a complete entity in the domain (e.g., bonds, elements, etc.). Our algorithm searches for candidate symbols (henceforth referred to as candidates) among groups of temporally or spatially contiguous strokes. It forms the set of temporal candidates by considering all possible sequences

Feature	Description
Length*	The length of the segment.
Ink Density	The length of the stroke region matching the segment divided by the length of the segment.
Segment Count	The total number of segments in the parent stroke (discrete, ceiling=10).
Stroke Diagonal*	The diagonal length of the parent stroke's bounding box.
Stroke Ink Density	The length of the parent stroke divided by the diagonal length of the parent stroke's bounding box.

Table 2. List of geometric features for segment classification. (*) means we include two version of this feature, one normalized by L and the other unnormalized.

Feature	Description
Stroke Count	The number of strokes in the candidate (discrete, ceiling=10).
Segment Count	The number of segments in the candidate (discrete, ceiling=10).
Diagonal*	The diagonal length of the candidate's bounding box.
Ink Density	The cumulative length of the strokes in the candidate divided by the diagonal length of the candidate.

Table 3. List of geometric features for candidate classification. (*) means we include two version of this feature, one normalized by L and the other unnormalized.

of up to $n = 8$ consecutively drawn strokes. It forms the set of spatial candidates by combining groups of strokes that are close to each other. This process starts with all possible groups of size 2 (each stroke and its nearest neighbor) and successively expands each group by including the next nearest stroke (e.g., each stroke and its 2 nearest neighbors, then its 3 nearest neighbors, etc.). This expansion ends when either the size of the group exceeds a spatial constraint or when the group contains more than 4 strokes. This spatial grouping algorithm allows temporal gaps in candidates, so symbols need not be drawn with consecutive strokes.

The features we use for candidates encode the visual appearance of the candidate, based on our previous work in [16]. For each symbol we generate a set of five 20x20 feature images, four orientation filter images and one "endpoint" image that captures the location of stroke endpoints. These feature images contain only the strokes that belong to the candidate (unlike feature images in the other levels, which include all the ink in a local patch). In order to improve robustness to differences in aspect ratio, we stretch each candidate symbol so that it has the same standard deviation of ink in both the x and y axes. As before, we smooth and downsample each image by a factor of 2. An example is shown in Figure 4. Notice that the "S" (candidate 2) is stretched horizontally to ensure equal standard deviation of ink in both axes.

In addition to these five feature images, we include another set of four images that describe the ink in a patch around the

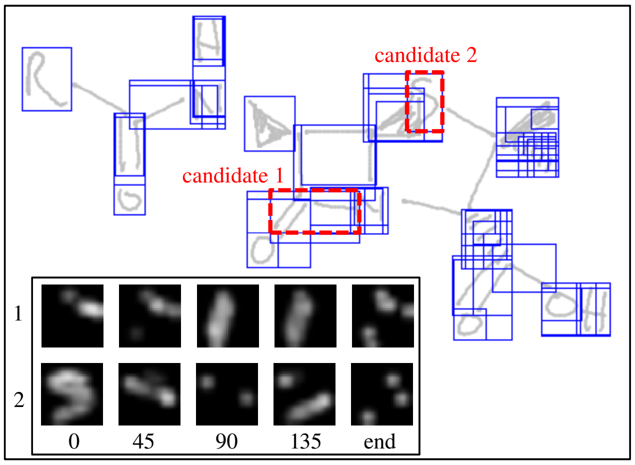

Figure 4. The set of candidates extracted from a chemical diagram, shown boxed in blue. The feature images generated for the two candidates highlighted in red are also shown below.

candidate. These are identical to those used for segments, but are centered at the center of the candidate with a region size of L. The result is a total of 600 feature image values. We also include as features the set of geometric properties listed in Table 3.

Feature Image Templates

In the sections above we described how the system generates sets of feature images for each classification entity (i.e., inkpoints, segments, and candidates). However, we do not use the image values directly as features for classification. Instead, we compare the images against a set of stored templates taken from the training data and record the match distances to the nearest template neighbor in each class. In order to make matches at the candidate level rotation invariant we test 8 evenly-spaced rotations of the candidate symbol. Next, we convert these distances into match scores (score = 1.0 - distance) and use as features both the label of the nearest neighbor and the best match scores to each class. For example, a candidate whose nearest neighbor is an "N" (Nitrogen) symbol might have the following features: (nearest="N", score.N=0.7, score.H=0.5, etc.).

To improve the speed and memory usage of the template matching process described above, we use principal component analysis to reduce the dimensionality of the feature images for each entity to 256. For example, we compress the 400 image values from an inkpoint to 256 principal components. We then calculate match distances based on these principal components rather than the original image values.

JOINT GRAPHICAL MODEL CLASSIFIER

We propose a new model for sketch recognition based on conditional random fields (CRF) that combines the features from the three levels in the classification hierarchy. A CRF can be seen as a probabilistic framework for capturing the statistical dependencies between the different entities we wish to model (i.e., inkpoints, segments, and candidates).

An alternative architecture is to train an independent classifier at each level, then use some type of voting scheme to combine the predictions. This approach has two disadvantages. First, by treating each layer in isolation it ignores any joint dependencies between features at different levels. Second, it requires the designer to specify a weighting scheme for each layer (e.g., deciding that the predictions in the candidate layer should be worth 2x those in the inkpoint layer) either manually or by some separate learning process.

Figure 5 shows an illustration of our CRF graph structure. The nodes in the bottom row represent labels for inkpoints (V_p), nodes in the middle row represent labels for segments (V_s). Inkpoint and segment nodes each have four possible labels: "bond" (straight bond), "hash", "wedge", and "text". The "text" label temporarily condenses the specific letters and abbreviations (e.g., "H", "O", "R") into a single label. When classification is finished, any candidate symbol recognized as "text" is converted back to the letter identity of its nearest template match.

The nodes at the top level represent symbol candidates (V_c). Notice that our model creates one candidate node for each segment rather than one for each candidate. This node contains, as possible labels, all of the candidates that the segment could belong to. During the inference process the system chooses the best candidate for each segment and adds the candidate to the set of final symbol detections. For example, if the system decides that the correct label for $y_{c,2}$ (the candidate node for segment 2) is a "wedge" candidate containing segments [1,2,4], then the "wedge" candidate is added to the final symbol detections. Note that the candidate node labels can contain multiple interpretations of each candidate, so $y_{c,2}$ also has "hash" and "text" versions of candidate [1,2,4] as possible labels (the "bond" label is only applied to single-segment candidates).

The edges in our CRF model encode four types of relationships between nodes:

Entity features to label mapping: We define ϕ as the local potential function that determines the compatibility between an entity's features and its label. This is analogous to a local classifier that classifies each entity independently of the others.

$$\phi_p(y_i, \mathbf{x}_i; \theta) = \sum_l f_{p,l}(y_i, \mathbf{x}_i)\theta_{p,l} \tag{2}$$

Here $\mathbf{x}_i$ is the set of features for entity i, y_i is the label, and f is a feature function defining the set of features for the given entity (e.g., the 256-valued PCA vector for inkpoints). There are three versions of this relationship: ϕ_p (shown above) for inkpoints, ϕ_s for segments, and ϕ_c for candidates (the same is true for f and θ). Note that ϕ is linear to the parameters θ, making the joint model (Equation 6 below) log-linear. In the case of candidate nodes the situation is more complicated because the labels can map to different candidates. Therefore the feature function here depends on the candidate of label y_i.

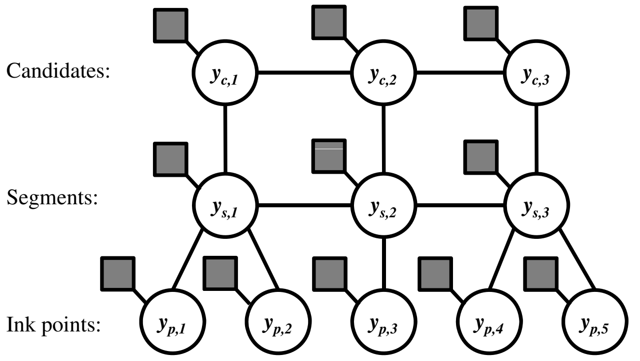

Figure 5. An illustration of our conditional random field model. Circles represent label nodes (y), edges represent relationships, and dark boxes represent evidence nodes (x) that connect the label nodes to their corresponding features.

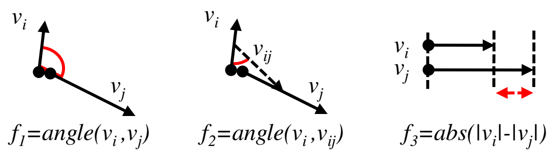

Figure 6. The three pairwise relationships used in the spatial context compatibility between segments.

Cross-level label consistency: This is a pairwise constraint stating that predictions at each level need to be consistent with predictions at other levels. An inkpoint and its parent segment should have the same label, and a segment and its parent candidate should have the same label.

$$\psi(y_i, y_j) = \begin{cases} 0, & \text{if } y_i = y_j \\ -\text{inf}, & \text{otherwise} \end{cases} \tag{3}$$

Segment to segment spatial context: This pairwise relationship captures the spatial compatibility between pairs of segments given their respective labels. This relationship enables our system to classify each segment jointly with its context, allowing neighboring interpretations to influence each other.

$$\psi_s(y_i, y_j, \mathbf{x}_i, \mathbf{x}_j; \theta) = \sum_l f_{ss,l}(y_i, y_j, \mathbf{x}_i, \mathbf{x}_j)\theta_{ss,l} \tag{4}$$

Here the feature function $f_{ss,l}$ contains the 3 spatial relationships shown in Figure 6. The system discretizes f_1 and f_2 into bins of size $\pi/8$ and f_3 into bins of size $L/4$.

Candidate to candidate overlap consistency: This is a pairwise constraint that prevents the system from choosing two different candidates that share any of the same segment(s), resulting in conflicting interpretations for those segment(s). For example, if the system decides that the label of $y_{c,2}$ (the candidate node for segment 2) is a "wedge" candidate that spans segments [1,2,4], then the labels for $y_{c,1}$ and $y_{c,4}$ also need to be assigned to the same "wedge" candidate.

$$\psi_c(y_i, y_j) = \begin{cases} 0, & \text{if } y_i = y_j \text{ or} \\ & y_i \text{ does not overlap } y_j \\ -\text{inf}, & \text{otherwise} \end{cases} \tag{5}$$

Combining all of the relationships described above, the joint probability function over the entire graph is:

$$\begin{aligned} \log P(\mathbf{y}|\mathbf{x}, \theta) = & \sum_{i \in V_p} \phi_p(y_i, \mathbf{x}_i; \theta) + \sum_{i,j \in E_{ps}} \psi(y_i, y_j) \\ & + \sum_{i \in V_s} \phi_s(y_i, \mathbf{x}_i; \theta) + \sum_{i,j \in E_{sc}} \psi(y_i, y_j) \\ & + \sum_{i \in V_c} \phi_c(y_i, \mathbf{x}_i; \theta) + \sum_{i,j \in E_{cc}} \psi_c(y_i, y_j) \\ & + \sum_{i,j \in E_{ss}} \psi_s(y_i, y_j, \mathbf{x}_i, \mathbf{x}_j; \theta) - \log(Z) \end{aligned} \tag{6}$$

where E_{ps} is the set of label consistency edges from ink-points to segments, E_{sc} is the set of label consistency edges from segments to symbols, E_{cc} is the set of overlap consistency edges from candidates to candidates, and E_{ss} is the set of spatial context edges from segments to segments. Z is a normalization constant.

Inference and Parameter Estimation

During training the system estimates the parameters θ in a maximum likelihood framework. The goal is to find $\theta^* =$ argmax $L(\theta)$, where, following the previous literature on CRFs [8], we define:

$$L(\theta) = \log P(\mathbf{y}|\mathbf{x}, \theta) - \frac{1}{2\sigma^2}||\theta||^2 \tag{7}$$

Here the second term is a regularization constraint on the norm of θ to help avoid overfitting. We optimize $L(\theta)$ with a gradient ascent algorithm, calculating the gradient for each parameter $\frac{\delta}{\delta\theta_i}L(\theta)$. This process requires us to compute the marginals $P(y_i|\mathbf{x}, \theta)$. Since loops in the graph make exact inference intractable, we calculate these marginals using loopy belief propagation [11], an approximate inference algorithm. We employ a randomized message passing schedule and run the BP algorithm for up to 100 iterations. For gradient ascent we use L-BFGS [12], which has been applied successfully to other CRF-based problems in the past [20]. We use the same belief propagation algorithm during inference.

Real-Time Recognition

Our system takes about 1 second to classify a sketch on a 3.7ghz processor running in a single thread. While this is likely sufficient for real time recognition, we also took steps to make sure that our system is fast enough to run on slower Tablet PCs. First, we implemented an incremental recognition model that updates the interpretation only of strokes and segments that have been modified or added since the last pass. Second, we made the most time consuming step of the process, generating features and template matches, parallel so that we could take advantage of multi-core CPUs. In our on-line user study a 1.8ghz Tablet PC was able to easily keep up with the users' drawings.

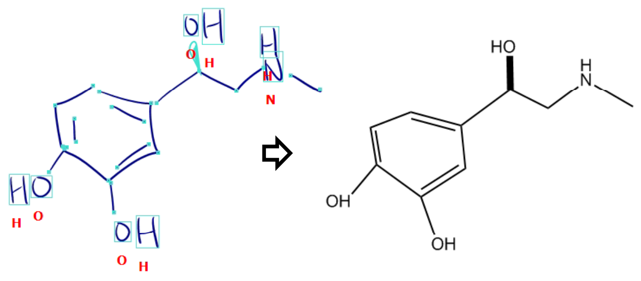

Figure 7. An illustration of the structure interpretation process: (left) an interpreted sketch with detected symbols highlighted and (right) the generated structure exported and rendered in ChemDraw.

STRUCTURE GENERATION

After choosing the final set of symbol detections, our system builds a connectivity graph between the symbols to produce the complete molecular structure. An example is shown in Figure 7. This symbol connectivity analysis is based on three pairwise distance metrics:

- Bond-element distance: The distance between a bond and an element is the distance from the bond endpoint to the nearest point in the element symbol. We impose an additional penalty if the bond does not point towards the element. For hash and wedge bonds, we define the direction of the bond as the principal axis based on PCA.
- Element-element distance: The distance between two letter symbols is defined as the distance between the two at their closest point.
- Bond-bond distance: The distance between two bonds is defined as the distance between their respective endpoints. We impose a penalty if the bonds do not point towards each other (e.g., if one bond is pointed to the midpoint of the other) or if they are nearly parallel (though parallel double bonds are technically connected to each other, we are interested in determining the elements to be joined at either of their endpoints).

We use an agglomerative clustering algorithm to generate the set of symbol connections. The algorithm iteratively merges the two nearest symbols or symbol clusters, using the maximum distance between the entities in the two groups as the clustering metric (i.e., complete-link). We empirically set the threshold to stop clustering at $0.4L$. Since as a general rule all symbols should be connected to at least one other symbol, the system reduces the distance value by a factor of two if there are only two symbols in the cluster. This encourages the algorithm to connect isolated symbols first and effectively lowers the threshold for single connections. We also impose a penalty if the cluster makes connections that violate the rules of chemical valence (e.g., connecting three bonds to an "H", as Hydrogen should form only one bond).

OFF-LINE EVALUATION

We recruited 10 participants who were familiar with organic chemistry and asked each of them to draw 12 real world organic compounds (e.g., Aspirin, Penicillin, Sildenafil, etc.) on a Tablet PC. We performed a set of user-independent performance evaluations, testing our system on one user while using the examples from the other 9 as training data. By leaving out sketches from the same participant, this evaluation demonstrates how well our system would perform on a new user.

Because one goal of our research is to build a system that can handle the range of drawings styles found in natural, real world diagrams, the program used to collect these drawings behaved simply like a piece of paper, i.e., capturing the sketch but providing no recognition or feedback. This ensured that the system did not inadvertently provide guidance in how to draw.

Corner Detection

We first evaluate the accuracy of our trainable corner detector in finding corners in bond strokes, where the corners determine the breaks between straight bonds. The results in Table 4 show that our algorithm was able to correctly detect 99.91% of the corners, with a precision of 99.85% (these measurements include stroke endpoints and single-segment strokes). In comparison, it outperformed a simpler version of the detector that uses a fixed threshold[2] on the cost metric from Equation 1.

Symbol Detection

The results in Table 5 show that our system was able to accurately detect and classify 97.4% of the symbols from the sketches in the dataset. Our result also represents an improvement on the best previously reported accuracy of 97.1% [15]. While the increase in performance seems modest, it is worth noting that performance on the dataset was already very high and may be beginning to plateau. Despite this, our new approach was able to remove over 10% of the remaining errors.

Note that for completeness we report precision as well as recall. However, for this task we believe that recall (the fraction of true symbols detected) is a more appropriate metric than precision (the fraction of detections that are true symbols) because, unlike in traditional object detection, there are no overlapping detections and every stroke is assigned to a symbol. Thus, a false positive always causes a false negative. Also, precision can be a less reliable metric because similar mistakes are not always counted equally. Misclassifying a 3-segment "H" as straight bonds, for instance, generates 3 false positives, while misclassifying it as a hash bond generates only one.

ON-LINE COMPARATIVE EVALUATION

We conducted a second user study to evaluate the usability and speed of our system, asking a number of chemistry graduate students to draw a set of five pre-selected diagrams on

[2]The threshold was chosen to produce similar values for the recall and precision.

Method	Recall	Precision
Trained detector	**0.9991**	**0.9985**
Fixed threshold	0.9879	0.9904

Table 4. Evaluation of the corner detection component of our system. We only count corners in strokes labeled as bonds and compare against the hand labeled ground truth.

Method	Recall	Precision
ChemInk (context)	**.974**	**.956**
ChemInk (no context)	.969	.951
O&D 2009 [15] (context)	.971	-
O&D 2009 [15] (no context)	.958	-

Table 5. Evaluation of the recognition accuracy of our system. The (no context) version does not employ spatial relationships between segments.

a Tablet PC[3]. While they were drawing, our recognition engine was running in real time and constantly providing feedback about the recognition progress by highlighting symbols detected so far. Users were asked to correct any errors the system made by simply erasing and redrawing the troubled region. Some of the study diagrams are shown in Figure 10.

We compared our system to an existing popular chemistry authoring tool called ChemDraw, asking users to produce the same diagrams using its traditional mouse-and-keyboard interface. We recorded each session and measured the amount of time taken to construct the diagrams using both interfaces. Afterwards we also asked the users for their opinions about how fast and easy it was to use each program.

Demographics

We had a total of 9 participants, all with prior experience with chemistry either through coursework only (1 user) or research only (1 user) or both (7 users). All of them had experience drawing chemical compounds on paper, reporting an average of 5.9 out of 7 (1=novice, 7=expert). Most also had extensive prior experience using ChemDraw, rating themselves on average a 5.0 out of 7. Conversely, most had little or no prior experience using Tablet PCs, rating themselves an average of 2.2 out of 7.

Quantitative Analysis

Figure 8 shows the average time that the users took to complete a diagram using both ChemInk and ChemDraw. It shows that they were on average more than twice as fast using our ChemInk sketching interface, averaging 36 seconds per diagram, compared to ChemDraw's average of 79 seconds. The difference in drawing time between the two interfaces is statistically significant (paired one-sided t-test, $p < .05$). This was a surprising finding for us since many of the participants mentioned that they had years of experience using ChemDraw and use it daily in their research. This finding would also likely surprise those users who did not rate ChemInk as being significantly faster in the subsequent survey (Figure 9).

[3]This study was conducted using an earlier version of our recognition engine, combining parts of our work in [15].

As Figure 8 shows, User 6 had an especially difficult time using ChemDraw, taking on average over 3 minutes per sketch. To make sure that the outlier was not biasing our results we repeated the analysis with User 6 omitted. The average time spent per sketch becomes 35 seconds for ChemInk and 61 seconds for ChemDraw, showing that our interface is still nearly twice as fast, and the difference is still statistically significant ($p < .05$).

Not surprisingly, the two users with the lowest prior ChemDraw experience (both rated themselves 1 out of 7) were also the slowest ChemDraw users (#6 and #8), and there was a highly negative correlation between reported ChemDraw experience and time spent per sketch (corr = -0.738, $p < .05$). This suggests that prior training is very important to using ChemDraw proficiently. In contrast, all of the users were able to use ChemInk effectively regardless of prior experience with Tablets PCs (corr = 0.111, $p > .05$).

Qualitative Analysis

On average users rated ChemInk as being faster (6.3 vs. 4.5) and easier to use (6.3 vs. 4.7) than ChemDraw (both differences are statistically significant, $p < .05$). In their comments about our system most of the users were very satisfied with the speed and performance, in many cases comparing it favorably against the traditional ChemDraw program. Some of the comments were: “Awesome!”, “The program was very good at recognizing my drawing even though I have fairly messy handwriting...”, and “In classroom setting, ChemDraw is too slow, whereas this is almost as fast as paper for taking notes.”

Some also had suggestions for making it faster: “It would be even faster if you have predrawn sections that you could insert...” and ”...if other letters or abbreviations were recognized, it would be even faster (for example recognizing that ph = [phenyl group])”.

One user made the interesting comment that there is something fundamentally different about sketching vs. click-and-drag: “I like drawing structures by hand rather than in ChemDraw. Just like w/ typing hand-written notes are easier to remember / visualize / understand. Ability to sketch in 3D is very important too. great work guys! :)”

RELATED WORK

There is a growing body of sketch recognition research covering a variety of different domains, including electrical circuits [4, 13, 19], general charts [22, 23], and mathematical expressions [9]. Many of these approaches focus on the relationships between geometric primitives like lines, arcs, and curves, specifying them either manually [1, 5, 6] or learning them from labeled data [19]. Recognition is then posed as a constraint satisfaction problem, as in [5, 6], or as an inference problem on a graphical model, as in [1, 19, 21, 23]. Similar to our approach, Szummer [23] proposed using a CRF to classify segments in a diagram by modeling the spatial relationships between neighboring segments. Their work differs from ours in that it focused only on segments and did not model complete symbols.

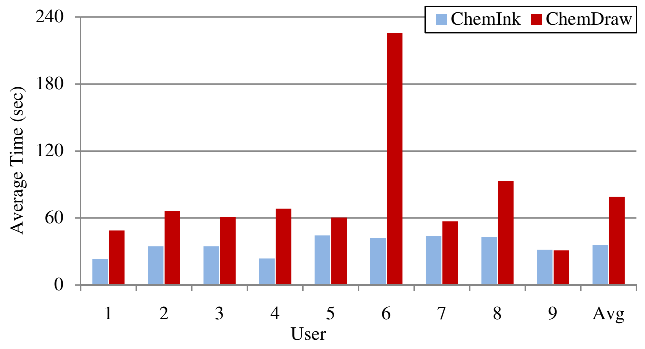

Figure 8. The average time taken by each of the study participants to draw a chemical diagram using ChemInk and ChemDraw.

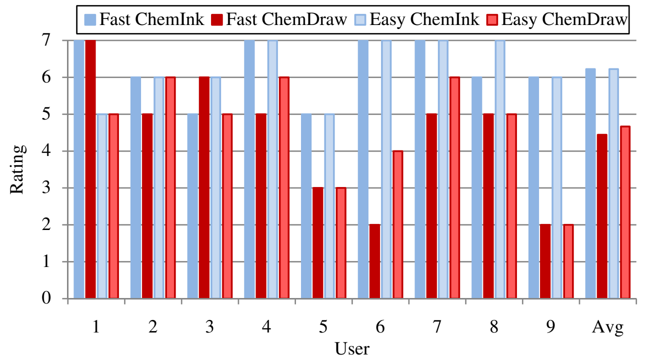

Figure 9. The ratings given by the study participants on how fast and how easy it was to use ChemInk and ChemDraw. Higher ratings indicate faster and easier.

Another group of related work focuses on the visual appearance of shapes and symbols. These include parts-based methods [13, 22], which learn a set of discriminative parts or patches for each symbol class, and template-based methods [7, 16], which compare the input symbol to a library of learned prototypes. The main advantage of vision-based approaches is their robustness to many of the drawing variations commonly found in real-world sketches, including artifacts like over-tracing and pen-drag. However, these methods typically do not model the spatial relationships between neighboring shapes, relying on local appearance to classify a symbol.

There have also been previous efforts to recognize chemical diagrams. Tenneson and Becker [24] developed a sketch-based system that helps students visualize the three dimensional structure of an organic molecule. Their system was able to avoid many of the challenges in sketched symbol detection by requiring that all symbols be drawn using a single stroke. It also did not handle implicit structure such as omitted carbon and hydrogen atoms. Casey et al. [2] developed a system for extracting chemical graphics from scanned documents, but their work focused on scanned printed chemical diagrams rather than freehand drawings.

This paper builds upon the work on multi-domain symbol detection from [14] and [15]. It presents a new principled ap-

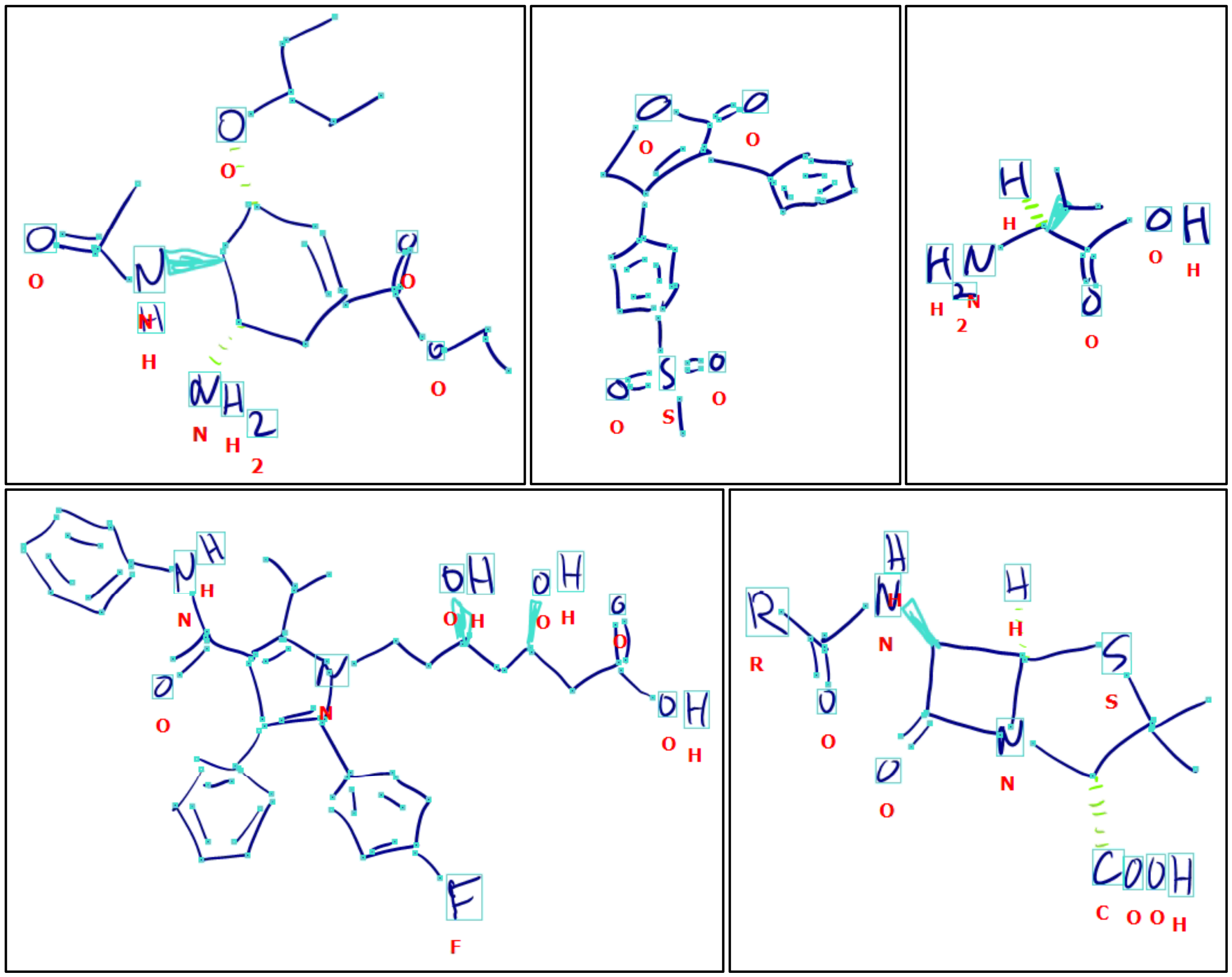

Figure 10. Examples of sketches collected from the on-line user study. The system's interpretation is highlighted as: text = boxed blue with letters below, straight bonds = blue, wedge-bond = light blue, hash-bond = green, bond endpoints = small boxes.

proach to combining multiple visual feature representations using a jointly trained CRF model. It also introduces a new learning-based approach to corner detection that achieves nearly perfect results in our evaluation.

GENERALITY OF OUR APPROACH

While this paper focused exclusively on chemical diagrams, we made efforts to design our approach to be as general as possible. As part of this choice, many components of our architecture expand upon previous techniques that have been proven in other domains. We have successfully applied the visual feature image descriptors, for example, to isolated symbol classification in electrical circuits, PowerPoint shapes, and handwritten digits [16]. Also, an earlier version of our graphical model approach was used in [15] to build a symbol detector for electrical circuit diagrams, improving upon the best existing benchmark for that dataset. While this does not mean that our architecture will generalize to other domains without modification, we believe that the results from these other studies are encouraging. This represents an exciting area for future work.

Our approach does make two important assumptions about the domain. First, it only detects symbols among temporally and/or spatially neighboring segments. Second, it assumes that segment breaks correspond to symbol boundaries, so that no segment belongs to more than one symbol. While we believe that these are reasonable assumptions for many domains (e.g., flow charts, circuits, course-of-action diagrams, etc.), if needed it should be possible to modify our algorithm so that it does not rely on these drawing patterns. For example, if there is no natural notion of segments or corners, the system could instead search for symbols among groups of inkpoints.

CONCLUSIONS

In this paper we introduced a new sketch recognition architecture for hand-drawn chemical diagrams. It combines rich visual descriptions of the sketch at multiple levels with a joint CRF model that captures the relationships between levels. Our system was able to correctly detect 97.4% of the symbols in a dataset of real-world chemical diagrams, improving on the best result previously published in literature. We also present a novel trainable corner detector that is accurate over 99% of the time. In an on-line study our new interface was over twice as fast as the existing CAD-based method for authoring chemical diagrams, even for novice users who had little or no experience using a tablet. While the evaluation is still preliminary, to our knowledge this is one of the first direct comparisons that shows a sketch recognition interface significantly outperforming a popular industry-standard CAD-based tool.

Acknowledgements
This research was supported in part by a DHS Graduate Research Fellowship, a grant from Pfizer, Inc., and NSF Grant Number 0729422.

REFERENCES

1. C. Alvarado and R. Davis. Sketchread: A multi-domain sketch recognition engine. In *Proc. UIST*, 2004.
2. R. Casey, S. Boyer, P. Healey, A. Miller, B. Oudot, and K. Zilles. Optical recognition of chemical graphics. *Document Analysis and Recognition*, pages 627–631, 1993.
3. L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding*, 106(1):59–70, 2007.
4. L. Gennari, L. Kara, T. Stahovich, and K. Shimada. Combining geometry and domain knowledge to interpret hand-drawn diagrams. *Computers & Graphics*, 29(4):547–562, 2005.
5. M. Gross. The electronic cocktail napkin - a computational environment for working with design diagrams. *Design Studies*, 17(1):53–69, 1996.
6. T. Hammond and R. Davis. Ladder: a language to describe drawing, display, and editing in sketch recognition. In *International Conference on Computer Graphics and Interactive Techniques*, 2006.
7. L. Kara and T. Stahovich. An image-based trainable symbol recognizer for sketch-based interfaces. *AAAI Fall Symposium: Making Pen-Based Interaction Intelligent and Natural*, 2004.
8. J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. ICML*, pages 282–289, 2001.
9. J. LaViola Jr and R. Zeleznik. Mathpad 2: a system for the creation and exploration of mathematical sketches. *ACM Transactions on Graphics*, 23(3):432–440, 2004.
10. D. Lowe. Object recognition from local scale-invariant features. In *International Conference on Computer Vision*, volume 2, pages 1150–1157, 1999.
11. K. Murphy, Y. Weiss, and M. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proc. of UAI*, pages 467–475, 1999.
12. J. Nocedal and S. Wright. *Numerical optimization*. Springer, 2000.
13. M. Oltmans. *Envisioning Sketch Recognition: A Local Feature Based Approach to Recognizing Informal Sketches*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 2007.
14. T. Ouyang and R. Davis. Recognition of hand drawn chemical diagrams. In *Proc. AAAI*, 2007.
15. T. Ouyang and R. Davis. Learning from neighboring strokes: Combining appearance and context for multi-domain sketch recognition. In *NIPS*, pages 1401–1409, 2009.
16. T. Y. Ouyang and R. Davis. A visual approach to sketched symbol recognition. In *Proc. IJCAI*, 2009.
17. B. Paulson and T. Hammond. Paleosketch: accurate primitive sketch recognition and beautification. In *Proc. IUI*, pages 1–10, 2008.
18. T. Sezgin and R. Davis. Sketch based interfaces: Early processing for sketch understanding. In *International Conference on Computer Graphics and Interactive Techniques*, 2006.
19. T. Sezgin and R. Davis. Sketch recognition in interspersed drawings using time-based graphical models. *Computers & Graphics*, 32(5):500–510, 2008.
20. F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *Proc. ACL*, pages 134–141, 2003.
21. M. Shilman, H. Pasula, S. Russell, and R. Newton. Statistical visual language models for ink parsing. *AAAI Spring Symposium on Sketch Understanding*, 2002.
22. M. Shilman, P. Viola, and K. Chellapilla. Recognition and grouping of handwritten text in diagrams and equations. In *Frontiers in Handwriting Recognition*, 2004.
23. M. Szummer. Learning diagram parts with hidden random fields. In *International Conference on Document Analysis and Recognition*, pages 1188–1193, 2005.
24. D. Tenneson and S. Becker. *Interpretation of Molecule Conformations from Drawn Diagrams*. PhD thesis, Brown University, 2008.
25. Y. Xiong and J. LaViola Jr. Revisiting shortstraw: improving corner finding in sketch-based interfaces. In *Proc. Eurographics Symposium on Sketch-Based Interfaces and Modeling*, pages 101–108, 2009.