ChemWARD: Extracting Chemical Structure from Printed Diagrams

by
Angelique Moscicki
S.B. CS, M.I.T., 2008

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology
September, 2009

Author. . . . .
Department of Electrical Engineering and Computer Science
September 7, 2009

Certified by . . . .
Prof. Randall Davis
Thesis Supervisor

Accepted by . . . .
Dr. Christopher J. Terman
Chairman, Department Committee on Graduate Theses

ChemWARD: Extracting Chemical Structure from Printed Diagrams

by

Angelique Moscicki

Submitted to the

Department of Electrical Engineering and Computer Science

September 7, 2009

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

Over the years, a vast amount of literature in the field of chemistry has accumulated, and searching for documents about specific molecules is a formidable task. To the extent that the literature is textual, services like Google enable relatively easy search. While search indexes like Google are very good at finding such things, its difficult to describe molecules completely using text because text can't easily indicate molecular structure, and molecular structure defines chemical properties. ChemWARD is a system that extracts the molecular structure from the printed diagrams that are ubiquitous in chemistry literature and converts them to a machine readable format in order to allow chemists to search the literature by drawing a molecular structure instead of typing a chemical formula. We describe the architecture of the system and report on its performance, demonstrating its ability to achieve an overall accuracy rate of 85.5% on printed diagrams extracted from published chemical literature.

Thesis Supervisor: Randall Davis
Title: Professor, MIT EECS

Acknowledgments

Ilya Baran, for contributing to the vectorization algorithm, and most importantly for being helpful even when it was difficult to be helpful.

The mathematicians at church, for their insights in geometry.

All the other people I have encountered who unknowingly turned an aimless wanderer into an engineer.

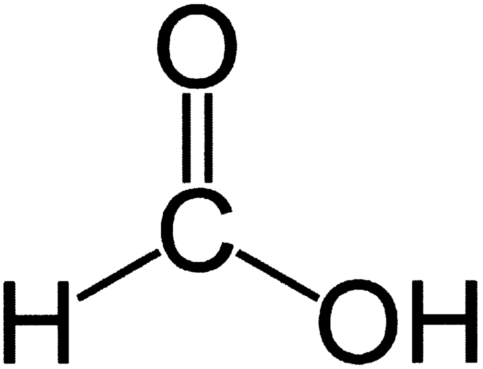
Pfizer Inc., for financially supporting this project.

Contents

Chapter 1

Introduction

Over the years, a vast amount of literature in the field of chemistry has accumulated, and searching for documents about specific molecules is a formidable task. To the extent that the literature is textual, services like Google enable relatively easy search. While search indexes like Google are very good at finding such things, its difficult to describe molecules completely using text because text can't easily indicate molecular structure, and molecular structure defines chemical properties. Molecular structure includes information about the geometry of the atoms, including such things as angles between bonds and the spatial arrangement of the atoms involved. It also includes topological information like the connectivity between atoms and the degree of each bond (i.e., a single bond, a double bond, etc.). It is far easier to represent this type of information in diagrams like the one shown in Figure 1.1 and such diagrams are ubiquitous in chemical literature.



IUPAC Name:	Formic Acid
Chemical Formula:	CH_2O_2
SMILES:	O=CO

Figure 1.1: Example of a diagram showing molecular structure, as is commonly found in chemistry papers. Below the diagram are several text representations of the molecule.

Textual descriptions of molecules are also present in chemistry papers, but the most common ones do not communicate information about molecular structure. The chemical formula, for example, just lists the number of each type of atom present in the molecule without specifying anything about connectivity. Another standard nomenclature, IUPAC, has the same problem and can in addition generate rather verbose names for molecules. SMILES is a text based representation that conveys topological information by representing atoms by their atomic symbols and bonds by other symbols like '='. SMILES, however, is not very human readable for large molecules with many ringlike substructures in them. It also contains no information about geometry.

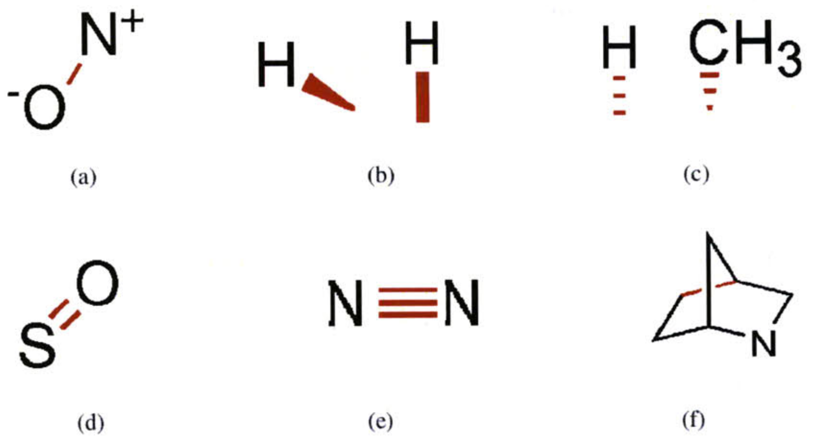
For all these reasons, the most complete description of a molecule that a trained human can understand at a glance is the structural diagram. The task of searching chemistry literature would be much easier if it were possible to search by drawing the structure of interest because the chemist would be spared the difficulty of having to convert to a less intuitive textual representation. Another advantage is that chemistry papers often include diagrams for molecules that are not explicitly named in the text. It would also enable chemists to search for partial structures as opposed to whole molecules.

Search by structure would certainly be a valuable tool, but what is required to make it possible? There are two parts to the problem; first it must be possible for chemists to quickly and easily create machine understandable structure diagrams that specify the molecule of interest. This is analogous to the task of typing the key word into the search box on Google. One particularly intuitive way to accomplish this is by using a sketch interface on a tablet PC; several systems have been constructed to do this [1] [2]. The other part of the problem is finding the diagrams in printed chemistry papers and interpreting them to retrieve the molecular structure they depict.

Extracting a complete description of a molecule from a diagram is sizable task. I break down into several subtasks. The first necessary step is to convert the chemistry papers to a digital format. Many new papers are already in digital formats, but older and historical documents exist only on paper, often with poor print quality. The pages of paper need to be scanned. The output of a scanner, however, is merely bitmap image data; it contains no information about whether the image contains text or numbers or a diagram or a photo, so the next step is to find the diagrams in the scanned pages and isolate them for interpretation. After the images have been found, they need to be interpreted to find the molecular structure, which is the focus of this work. Finally, the molecular structures must be integrated into a search index that chemists can make use of. The search index, combined with the sketch interface mentioned earlier would enable chemists to search for molecules by their structure.

This brings us to the central theme of this paper, which is to extract the molecular structure from chemical diagrams represented as bitmap images, which I will call the molecule recognition problem. After having read this thesis, you will understand in detail the challenges associated with the molecule recognition problem, and how to build a system to solve it. In order to make the task a bit better defined, I will now present a brief overview the anatomy of a molecular structure diagram.

Structure diagrams obey a standard set of conventions when depicting atoms and bonds. Atoms are represented straightforwardly by their atomic symbols. Sometimes diagrams will also contain standard chemical shorthand for simple common molecules that are part of the substructure, like 'Me' for a methyl group. Ions are indicated by a plus or minus superscript next to the atomic symbol. Aside from atomic symbols, structure diagrams also contain bonds. Single, double, and triple bonds are typically drawn as one, two, or three straight lines, respectively, between the atoms they connect. Sometimes, however, depictions of bonds are slightly more complicated due to the fact that molecules have a three dimensional structure that must be represented on paper. Conventions exist for drawing bonds that extend into the plane of the paper away from the viewer, and out of the paper towards the viewer. These are called hash bonds and wedge bonds, respectively, and are shown in Figure 1.2. Sometimes it is not possible to draw a molecular structure on paper without bonds crossing each other, so chemists express this by drawing split bonds, also shown in Figure 1.2. Atomic symbols, common shorthand, and they types of bond depictions shown in Figure 1.2 together make up the graphical vocabulary used to draw molecular structure, and consequently determine the types of symbols any system attempting to solve the molecule recognition problem must understand. Understanding the symbols in structural diagrams is a challenging problem that can itself be broken down into three general components:



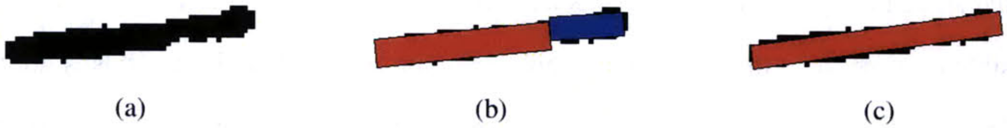
(a) (b) (c)

(d) (e) (f)

Figure 1.2: Shown in red are common ways of drawing bonds. (a) is a single bond, (b) shows single bonds extending out of the paper towards the viewer, (c) shows single bonds extending out of the paper away from the viewer. (d) is a double bond, (e) is a triple bond, and (f) is a single bond that crosses another single bond, so it is drawn in two pieces.

Recognizing Letters: The problem of recognizing text in images, or optical character recognition (OCR) has been studied since the 1960s [3]. Some high performance solutions have been developed in industry such as ABBYY FineReader, ReadIris and others. But, conventional OCR engines are not well suited to the molecule recognition problem because they are designed to read long blocks of text arranged in lines, of the type that would be found in a novel, and not the sparse, isolated characters found scatted throughout molecular diagrams. Superscripts and subscripts also present problems for conventional OCR engines because they tend to be small and difficult to read.

Recognizing Bonds: Because bonds are drawn as a line or a collection of lines, the problem of recognizing bonds is intimately related to the task of locating lines in the bitmap data. Regions of pixels that appear to be lines in structure diagrams are often noisy rectangular shapes at the pixel level, like the single bond shown in Figure 1.3 (a). Noise effectively makes it impossible to get a perfect correspondence between the pixels and a set of lines, so some amount of approximation is necessary. Approximating a bitmap image with a set of lines is also an old problem, called vectorization. As was the case with OCR, there exist vectorization algorithms that produce aesthetically beautiful results, but once again the type of output they generate is not well suited to the molecule recognition problem. This is because most vectorization algorithms seek to approximate the pixels of the image as accurately as possible, while in the molecule recognition problem, the real goal is to recognize structure. Complete and accurate pixel coverage should be sacrificed when it conflicts with extracting molecular structure. In particular, in a noisy image a region of pixels that a chemist understands to be a single line because it is a single bond ought not be approximated as two lines even if two lines more accurately cover the pixels.



(a) (b) (c)

Figure 1.3: (a) is a noisy set of pixels that is really a single bond. It takes two rectangles to cover all foreground the pixels while covering as few of the background pixels as possible, as shown in (b). Because (a) is really a single bond, the desired outcome is to approximate the region with a single rectangle like (c), even if the pixel coverage is not as good as in (b).

Handling Noise: Both poor print quality and poor scanning introduce noise into the bitmap data. Some examples are shown in Figure 1.4. Letters often are merged with other letters and bonds, bonds are merged with other bonds. In particular, when letters and bonds merge, the resulting shape resembles neither a letter nor a bond, so some type of segmentation technique must be employed.

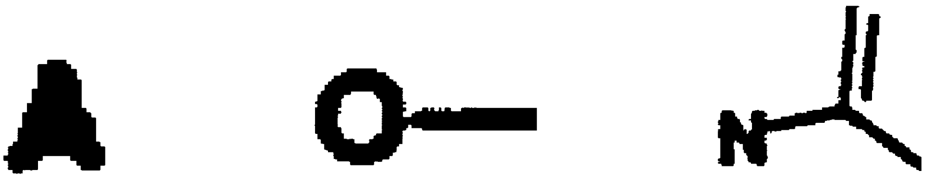


Figure 1.4: Examples of how noise can corrupt a diagram. The leftmost symbol is the letter 'A', the others show how noise can connect characters and bonds, a particularly difficult problem to address.

As mentioned earlier, search by structure requires both an interface for chemists to enter diagrams and a method to find, recognize, and index the diagrams already existing in the literature. In particular, the molecule recognition problem must be solved by both the interface for chemists and the program that processes diagrams in the literature. On the surface it may seem that since high performance systems already exist to solve the interface problem, it should be possible to reuse them to solve the molecule recognition problem. In fact this is not the case; recognizing molecular structure entered via a sketch interface is a very different problem from recognizing molecular structure in print. These problems are so different first because in a sketch interface there is a notion of temporal data that allows the system to separate the user input into lines and symbols according to when the pen is active. Information like the fact that users usually draw single bonds as a single pen stroke can be effectively exploited to make the recognition task easier. Sketch interfaces also have their own host of problems that do not occur in print, most notably the unpredictable user. Finally, sketch interfaces must run in real time in order to be useful, printed diagrams can be processed offline, and the processing need only be faster than a trained human in order to be useful. The problems are thus different enough that techniques that have been successfully used in sketch recognition can not be used with diagrams from printed sources.

The remainder of the chapter focuses on the approach that ChemWARD employs to solve the molecule recognition problem. One of the key ideas it uses is that interpretation occurs at multiple stages on the way from pixels to molecular structure.

Pixels must be grouped into sets that correspond to meaningful symbols.

- Symbols must be recognized as either letters or bonds.
- Connectivity between symbols must be established.
- Chemical knowledge can be brought to bear to ensure that the resulting graph of symbols is chemically valid.

While there is a certain degree of modularity between these levels of interpretation, they are also often related, with information at higher levels used to correct mistakes at lower levels. In ChemWARD, specific recognition tasks are delegated to special interpreters called "senses" that each operate only at the level or levels appropriate to their task, but can provide suggestions or corrections to other senses. For example, a sense responsible for checking that a particular atom does not have more bonds than chemically makes sense should operate on graphs, not pixels. Similarly, in order to keep complexity manageable, a sense responsible for OCR should not be concerned with bonds connected to the character it is recognizing. At the same time, it should be possible for high level senses to correct mistakes that low level senses made because they did not have enough information at the time they were first applied. Just as human perception is influenced by what we expect to perceive, ChemWARD uses high level reasoning to guide the more generic low level reasoning. Figure 1.5 illustrates an example of this type of reasoning.

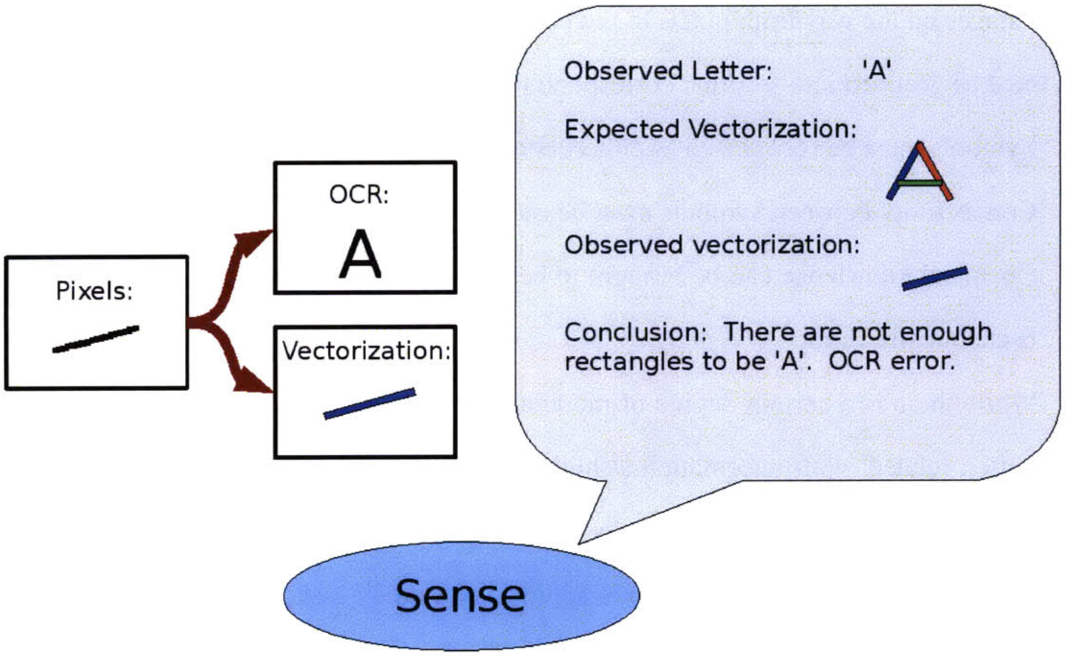


Figure 1.5: Example of how a high level sense that considers both vectorization and OCR can correct mistakes in the lower level OCR.

The idea of senses influencing each other in this way is described in more detail by Coen [4], and even earlier in the blackboard metaphor [5]. It is the guiding philosophy in ChemWARD. To summarize Coen's idea, in classical artificial intelligence, perception systems have senses built in a linear fashion, where each successively higher sense builds only upon the previous one. This forces the system to commit to decisions early on with too little information and does not allow for making corrections later when more information comes to light. It also clearly demonstrates another guiding principle in ChemWARD: it is better to remain uncommitted to an interpretation and retain ambiguity than it is to commit and allow mistakes to propagate. The classical linear sense architecture also reduces the efficacy of high level senses because they have access only to whatever the next lowest sense provided them; all intermediate representations are destroyed and can no longer be leveraged. According to Cohen, perception should not be a system where information flows strictly upwards. Downward information flow and horizontal information flow should also take place. Figure 1.6 shows a perception system according to Cohen's scheme of influence networks. ChemWARD consists of a similar network of senses that bootstrap off each other and correct each other in order to reach a consistent final interpretation of the molecular structure in a diagram. How each of the senses work, and how they interact, is described in detail in the rest of the thesis.

Chapter 2 describes the architecture and design philosophy of ChemWARD, and establishes the principles common throughout the entire system. Look here to see a zoomed out view of the entire system. Chapter 3 focuses on OCR, what problems I encountered when working with traditional OCR engines, and the steps ChemWARD takes to correct the inevitable OCR mistakes even the best engines will make. Grouping letters together into atomic symbols, chemical shorthand, and chemical formulas that appear as part of the overall molecular structure is the subject of chapter 4. Chapter 5 discusses vectorization and presents ChemWARD's novel algorithm that is designed to overcome the shortcomings of traditional ones. After that, chapter 6 describes the senses that refine the vectorization output, which due to noise is necessarily imperfect. Here ChemWARD makes extensively uses drafting knowledge about how structural diagrams are drawn. Chapters 3 and 4 can be read independently of Chapters 5 and 6. Chapter 7 covers how chemical knowledge is used to resolve inconsistencies in the ChemWARD's final output. Limitations of the current incarnation of ChemWARD are discussed in chapter 8, along with the direction of future work. To date, there are several other systems that attempt to solve the molecule recognition problem, including CliDE [6][7], ChemOCR[8], and OSRA[9]. Results and comparison with existing systems is the subject of Chapter 9. Finally, Chapter 10 summarizes the contributions ChemWARD has made to the molecule recognition problem.

Chapter 2

Architecture

In this chapter, I describe the architecture used in ChemWARD and the decisions that led to it. First I point out a few principles used throughout, and then I present some examples to show where simple approaches fail and to provide motivation for the architecture that I chose. I describe the stages of interpretation that ChemWARD operates on, and the intermediate representations that ChemWARD constructs while working toward a final understanding of the molecular structure in a given diagram. Finally I describe the representation that ChemWARD uses to cope with the specific challenges that arise as a result of using influence networks.

Several ideas are so central to the design used in ChemWARD that they should be kept in mind throughout the rest of this thesis. These are:

- Use high level information to correct low level mistakes.
- The further a mistake propagates, the more difficult it is to correct.
- It is better to remain uncertain about an interpretation and maintain ambiguity than to commit early to a mistake.
- It is easier to rule out possible interpretations of a symbol than to commit to one.

A concrete example will help motivate the architecture used in ChemWARD. Let us assume for the moment that there is very little noise in the input image and there are only single bonds in the plane of the paper (no hash, wedge or split bonds). Figure 2.1 shows such a molecule. Let's walk through the steps needed to interpret it.

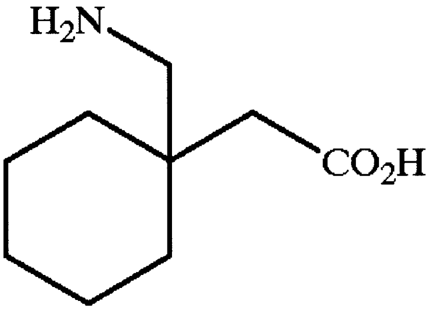


Figure 2.1: A very simple molecule entirely in the plane of the paper.

Let's start by assuming that every connected black region of pixels corresponds to an interesting symbol like a letter, a bond, or a group of bonds. In Figure 2.1 there are seven of these connected regions that are letters or digits that make up the formulas in the molecule, and one that is a large group of bonds. Let us also assume that we have at our disposal an OCR sense that proposes one or more alphanumeric guesses for what a connected region of pixels might be, and a vectorization sense that approximates a connected component with a set of rectangles. In a high quality vectorization, each rectangle will correspond to a single bond. Figure 2.2 shows the type of output each of these senses produces.

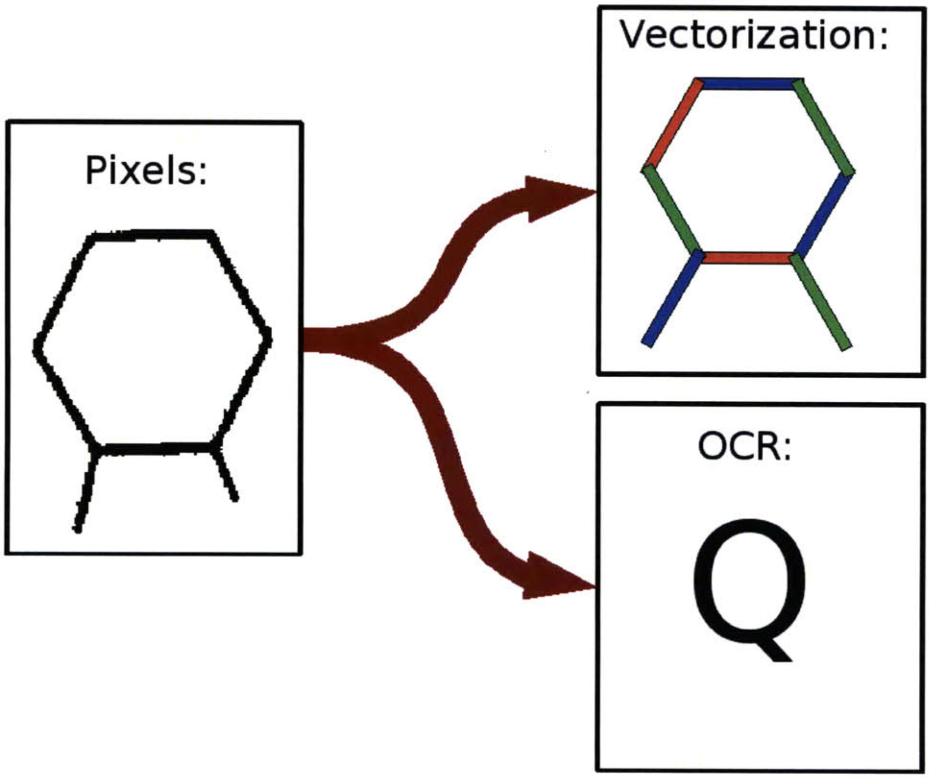


Figure 2.2: On the left is the pixel input, on the right are the types of output from the vectorization and OCR senses, respectively.

At this point we can classify the symbols of interest into two broad categories: alphanumeric symbols, and bonds. But which type should we try to identify first?

Let's first try applying OCR to every connected region of pixels, and trust that it will pick out all the letters in the image. The remaining connected regions must then be bond groups, and the vectorization sense should be able to pick out the rectangles that correspond to the individual bonds in each one. As described in the previous chapter, however, OCR does not perform as well as hoped. Figure 2.3 shows some mistakes that ChemWARD's unaided OCR sense made in real images. Even if OCR were designed to perform well in a domain like molecular structure diagrams, bond groups in the diagram sometimes legitimately look like letters when considered out of context, as shown in Figure 2.2 with the 'Q'. Hastily classifying them as letters would be a mistake, and mistakes are harder to correct the farther they propagate.

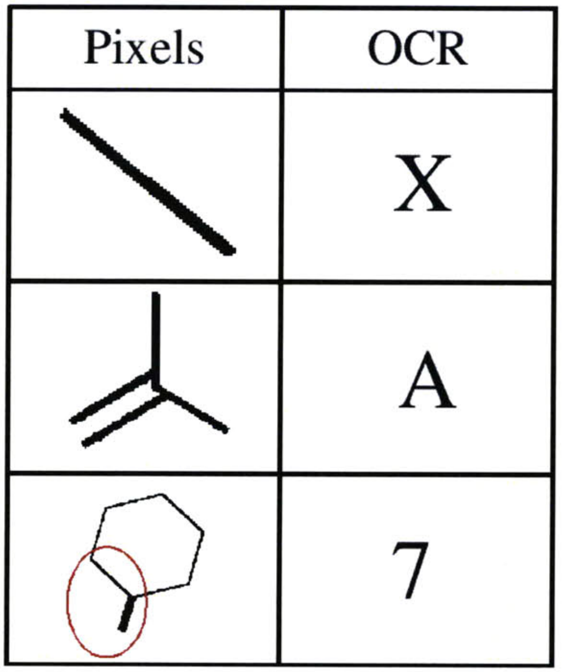


Figure 2.3: Examples of OCR mistakes. Note in the third row how the circled feature of the bond group might appear to be a '7'.

If applying the OCR sense first didn't achieve a very good result, lets try the opposite approach: use the vectorization sense first and try to pick out the bonds. Apply OCR only to those connected components whose vectorization contains many relatively small rectangles, or exhibit some cues to indicate that the vectorization is poor according to some metric we can devise. Figure 2.4 shows the problem with this approach; that even though the 'N' is distinguishable by the arrangement of the rectangles that form its vectorization with respect to each other, it is not distinguishable by any information in the vectors themselves. In order to recognize the 'N', contextual information has to be leveraged, and at such a low level as vectorization, this information has not been computed yet. The vectorization in Figure 2.4 is nearly perfect; in images with more noise and consequently worse vectorizations, the arrangement of rectangles becomes an unreliable indicator as to whether or not a connected region is a letter, so OCR will have to be used anyway.

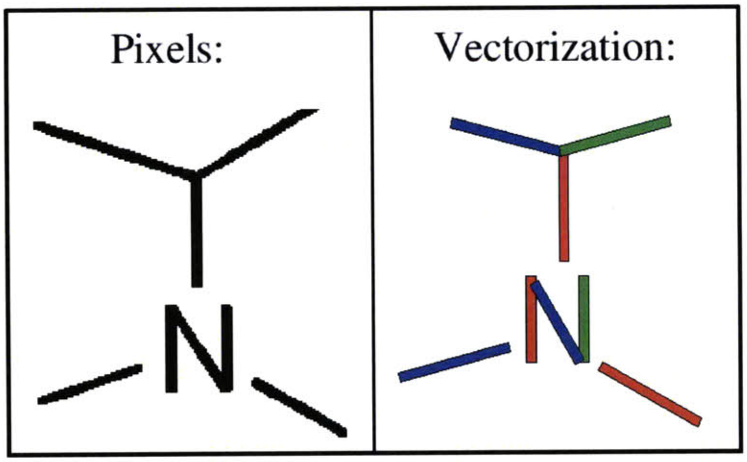


Figure 2.4: The vectorization on the right is very accurate, but the rectangles in the 'N' are the same size as the bond rectangles around them. There is no way to determine which rectangles are bonds, and which are the 'N'.

The message to take away from this example is that even with very little noise and only the simplest chemical symbols, there is enough genuine ambiguity in molecular structure diagrams to require a more complex model than a linear application of senses. An influence network would not suffer from this handicap because information can be shared across senses. But in order to build an effective influence network, senses must be able to extract from the image cues that are truly indicative of the molecular structure, and that do not try to do too much at once. For example, it might be a bit of a leap to assume that every rectangle produced by the vectorization sense really is a single bond, especially when hash, wedge, and split bonds are considered.

With this in mind, I now present an overview of the stages of interpretation used in ChemWARD. Each stage of interpretation is an intermediate representation somewhere between pixels and molecular structure, and has one or more senses responsible for producing it. Each one summarizes the state of the knowledge computed so far, and adds to it. After I've defined the stages of interpretation, I show how the senses associated with each one interact to form an influence network. The stages of interpretation are:

Pixels: Pixel level operations are vectorization and OCR.

Vectors: Because the vectorization problem was first explored for converting bitmap images to vector graphics, the rectangles produced by a vectorization algorithm are called vectors. The algorithm used in ChemWARD identifies both rectangles, and circular annuli. Once ChemWARD has vectorized an image, it uses knowledge about drafting to refine the vectorization by eliminating or merging vectors that seem spurious, grouping sets of vectors that form specific shapes together into vector sets, and by identifying regions that are possibly letters.

Characters: After a preliminary pass of OCR, its possible to roughly estimate the height and width of a character and eliminate mistakes like the 'Q' from Figure 2.2. At this stage, ChemWARD also aggregates characters together into groups that make up chemical formulas that are part of the overall chemical structure, as shown in Figure 2.5.

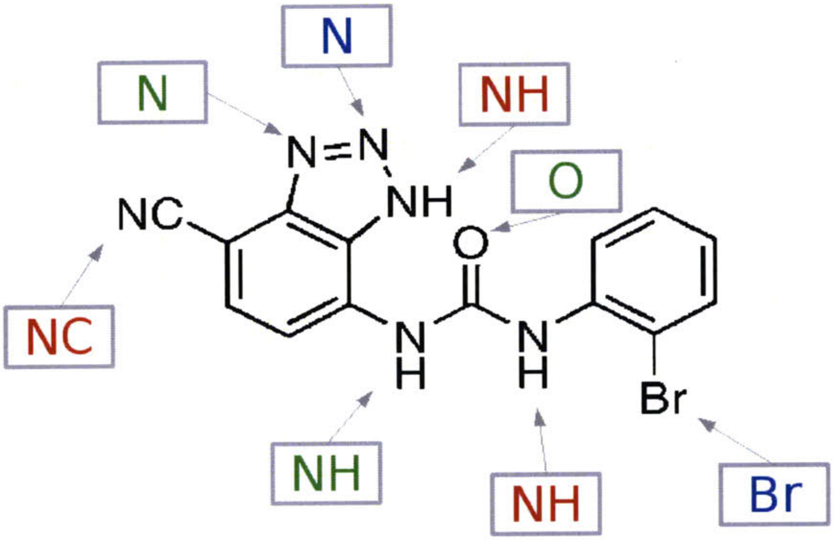


Figure 2.5: After the OCR sense identifies a potential set of characters, ChemWARD aggregates them into groups that represent chemical formulas that are part of the molecular structure. Notice how sometimes chemical formulas are written vertically instead of the usual horizontal left to right way.

Vector Topology: After the vectors have been refined, ChemWARD overlays a graph on top of them to indicate which vectors are adjacent to each other. Adjacent in this context means that the vectors have overlapping area. Every pair of adjacent vectors are connected in the graph.

Vector Sets: Because some chemical symbols like split bonds and hash bond are made of more than one vector, it is more appropriate to think of bonds as sets of vectors (and in the case of a standard single bond the set has only one vector in it). These sets can be converted into a bond representation where the notions of bond degree (i.e., single, double, etc.) and stereochemistry (i.e., whether the bond is in the plane, extends out of the plane, or into it) are added.

Formulas: After characters are grouped together into formulas, the formulas must be checked for syntactical errors. While some formulas are ambiguous, it is often possible to calculate how many bonds to expect. For example, a Nitrogen atom ('N') usually supports 3 bonds, where the formula C_3H_8O (isopropyl alcohol) is ambiguous because there are multiple isomers of the formula.

Chemical Topology: At this point enough low level interpretation has taken place to construct the graph that represents the actual molecule. Vector sets that have been converted into bonds and characters that have been aggregated into chemical formulas are connected to make a chemical graph that describes both the geometry and the topology of the molecule. Only now can the highest level senses check for consistent valence information, logical bond connectivity, etc. When such a sense discovers an inconsistency it can either modify the chemical graph to repair the problem, or point out the problem to the user for manual correction.

These are the intermediate knowledge representations ChemWARD uses. Figure 2.6 shows them as part of ChemWARD's influence network, and also shows the senses that produce, examine, and modify them. Subsequent chapters describe in detail each part of the network in Figure 2.6. Looking back at the guiding principles used in ChemWARD, the influence network provides several advantages. First, it clearly shows that high level senses can correct low level mistakes, as the Character Aggregator can be used to correct noise in the Pixels. Secondly, it facilitates the ability to avoid committing to a particular interpretation because as more information comes to light, senses can hand off data to one another. Thirdly, because high level senses can make changes downward, mistakes can be corrected at their source, and the corrections can bubble upwards through the usual processing pathways. For example, in Figure 2.6, the Vector Set Sense might correct a mistake in the Vectors, which triggers the Vector Sense and the Vector Graph Builder to re-examine the modified region and make their own changes in light of new information. Finally, the Vector Set Sense can observe the effects of the modification that it made, and possibly make further modifications. Throughout the rest of the thesis, there will be many concrete examples of precisely this behavior.

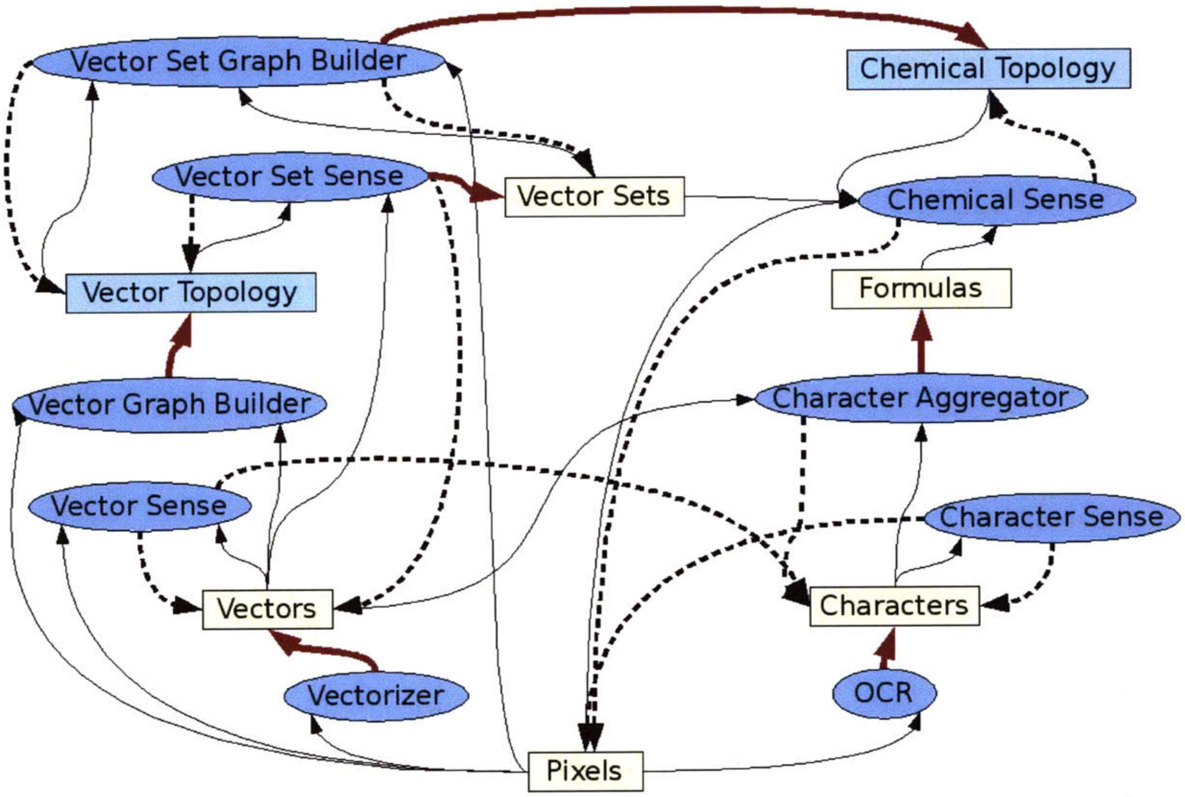


Figure 2.6: Influence network used in ChemWARD. Each oval is a sense, each rectangle is an intermediate knowledge representation. Green rectangles identify topological information whereas yellow ones identify classification information. The heavy red arrows trace the flow of information from pixels to chemical topology, from which the molecular structure is simply read off. The heavy dashed arrows trace the modifications that senses make to lower level intermediate representations. The thin black arrows identify the intermediate representations that each sense uses as input.

On the other hand, managing complexity that results from this and any other substantial influence network is a challenging task. It is critical that the intermediate representations remain consistent with one another, and there must be a mechanism for resolving conflicts between senses. In order to deal with these concerns, ChemWARD uses a special representation, called a column.

A column is a mechanism to propagate the changes made by one sense to the data structures used by all the other senses. As more and more senses process the image, each intermediate representation described above is computed. That means that each connected region of pixels is both vectorized and run through the OCR sense. On top of these layers, the vector graph (that indicates which vectors are adjacent) and the chemical graph (that reflects the chemical structure) are formed. Inconsistencies can easily arise between the layers; for example when a chemical sense finally commits to interpreting a region of pixels as a character and has to remove any vectors associated with the region from the vector graph so senses operating on the vector graph don't inadvertently identify any of the vectors computed earlier for the region as a bond. But, a high level sense should not need to operate on low level representations; semantically the low level representation is unintelligible to it, and practically it breaks the very abstraction barriers put in place to reign in complexity. Additionally, for each intermediate representation there are only a few types of permissible modifications. For example, a Vector Set supports only adding and removing vectors as possible modifications to it. Instead of having each sense explicitly worry about how the modifications it makes at the level it is built to reason at may affect representations above and below, it is better to have all the senses interact with the data via an intermediary that knows how to propagate all the changes up and down. This intermediary is the column. When a sense modifies the top layer, the column propagates up and down all the changes in a way appropriate to each intermediate representation. The column also specifies concrete interfaces by which senses are allowed to modify data, and these interfaces can be turned on and off. A visualization of a column is shown in Figure 2.7.

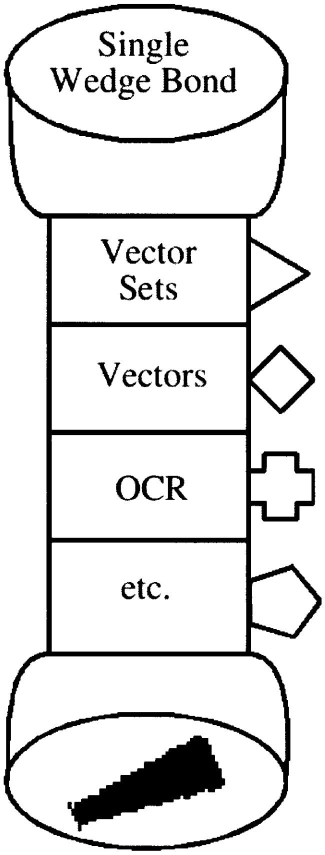


Figure 2.7: Conceptual diagram of a column. At the bottom is a region of pixels,at the top is the final interpretation of the pixels. In the middle are some of the interfaces that the senses are able to access when considering that region of pixels. Changes made via one interface are reflected in the data exposed at all of the interfaces, the column internally propagates the modifications.

With the problem of propagating data changes handled, there remains the task of resolving conflicts when senses disagree about the interpretation of a region of pixels. This is where the fourth guiding principle in ChemWARD becomes important: It is easier to rule out possible interpretations than to commit to one. Senses are allowed to generate their own interpretations for pixel regions and the column keeps track of them all. Other senses primarily reject interpretations until no more computation can be performed. Pixel regions for which all possible interpretations are eliminated are rejected as noise. If at the highest level there are still multiple interpretations, the Chemical Sense is the final arbiter in choosing among them. The best example of ambiguity that can be resolved only by the Chemical Sense is an Iodine atom in a molecule. In sans-serif font, the capital 'I' symbol for Iodine becomes a single rectangle. Figure 2.8 shows an example. It is impossible to determine whether the symbol is an 'I' or a single bond joining two chemical formulas by looking at the symbol in isolation. The deciding information is the presence of chemical formulas on the top and bottom of the symbol on the right and the lack of them on the left (the single bond connected to the 'I' on the left side of Figure 2.8 is not a reliable indicator because it does not actually touch the 'I').

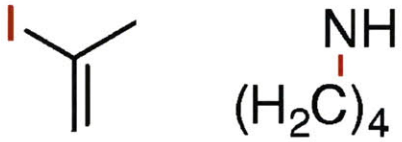


Figure 2.8: Two possible interpretations of a single rectangle. On the left, it is an Iodine atom as context indicates. On the right, it is a single bond.

In this chapter, I presented the guiding principles used in ChemWARD and the influence network that ChemWARD uses. I've described the different stages of interpretation that are important in understanding molecular structure and presented a representation that allows ChemWARD to successfully manage the complexity that influence networks can create. Over the next few chapters, I will focus on specific parts of Figure 2.6 and explain how each of them works.

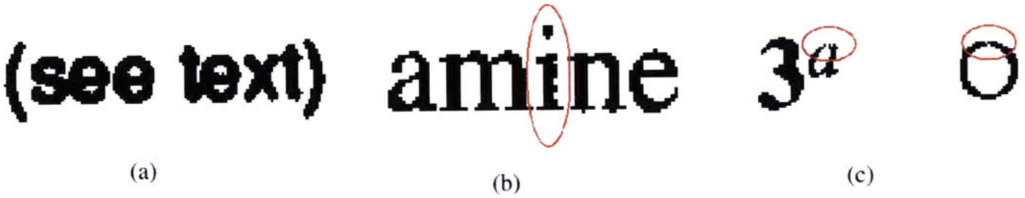
Chapter 3

Optical Character Recognition in ChemWARD

This chapter discusses the optical character recognition (OCR) subsystem used in ChemWARD. I begin by explaining more precisely what the OCR problem in general is and some of the challenges associated with it. Then I present some brief background information about solutions to the OCR problem and the state of the art today. I then explain how OCR in molecular diagrams is different from the general OCR problem and show examples of how traditional OCR engines failed on diagram input. Finally, I explain how ChemWARD solves the OCR problem.

Optical character recognition is the task of translating written, printed, or digital image text into editable text of the sort produced by word processors. While early OCR machines were truly optical in that they operated by shining light through slits onto printed documents, most applications today involve digital images. OCR has proven to be a difficult task for many reasons:

Noise: Digital images containing text come from two sources; either they are scans of printed documents or they are produced entirely on a computer but are exported into a format that does not preserve the text as such, e.g. image objects embedded in PDFs. The process of converting the original text into a digital medium in both cases can introduce noise from sources ranging from dirt on the scanner to a lossy image format. Some example of noisy characters are shown in Figure 3.1. As you can see, noise can alter both the appearance and the topology of a letter. The most common type of noise involves spurious dark pixels, sometimes merging characters together. Other problems include breaking a single letter into multiple pieces, and noise that makes the outlines of the letters fuzzy.



(a) (b) (c)

Figure 3.1: Types of noise in images of text. (a) shows characters merged with each other. Also note how noised joined the tail to the top half of all three examples of the letter 'e'. (b) shows how the edges of letters can become ragged. Letters broken by missing pixels are shown in (c).

Font: The same letter can look very different in different fonts. For example both ℰ and E are examples of the letter 'E', but the appearance is radically different. Whether or not the font has serifs has huge implications in this regard. In order to make the font more aesthetically pleasing, many fonts also intentionally join characters together using ligatures(Figure 3.2).

Figure 3.2: Font ligatures

Ambiguity: In some fonts, particular characters become nearly or totally indistinguishable. In many sans-serif fonts of the type commonly used in molecular structure diagrams, the characters upper case 'I' and lower case 'l' are identical. Similarly, zero can be nearly identical to upper case 'O'.

In spite of all these problems, researchers took an early and eager interest in OCR, and by the 1990s a prodigious number of techniques had been developed, including matching the characters to templates, analyzing the structure of the image data, and applying finely tuned neural nets. A very complete and detailed survey of the history of OCR can be found in [3]. While there is still some research on the problem in academia, the current state of the art in OCR is found in industry. Among the current top performers are FineReader [10], OCR Xpress [11], and ReadIris [12]. There are also open source OCR engines available [13][14], but they do not meet the same standard of performance as the commercial engines.

Today's OCR engines work best when faced with the task of converting images of documents with large amounts of text into machine editable ones. They can read most fonts, can often identify non-text regions of the image, and even preserve formatting and layout. In order to improve accuracy, many of them take advantage of contextual information like the language the document is written in. This allows the use of a dictionary to correct mistakes. As mentioned, OCR engines use the fact that text is arranged in lines on the page. They may also employ clustering techniques that can learn about the appearance of letters from numerous examples in the image. Under these favorable conditions, FineReader claims an accuracy rate of 99.995%; other commercial OCR engines are competitive with it.

Unfortunately, such conditions do not exist in molecular structure diagrams. In the first place, the characters in molecular structure diagrams are sparse and scattered, so there are no reliable lines of text to latch onto, presenting problems for OCR engines. In addition, chemical formulas in molecular structure diagrams are sometimes they are drawn top to bottom (Figure 3.3). The sparsity of characters also means that there aren't enough characters to use clustering techniques successfully. Constraining the allowable combinations of alphanumeric characters to those found in a dictionary would cause mistakes in this case rather than prevent them, because there are very few restrictions on what values the subscripts in a chemical formula can take. Finally, molecular structure diagrams contain many superscripts and subscripts that are particularly susceptible to noise because they are so small.

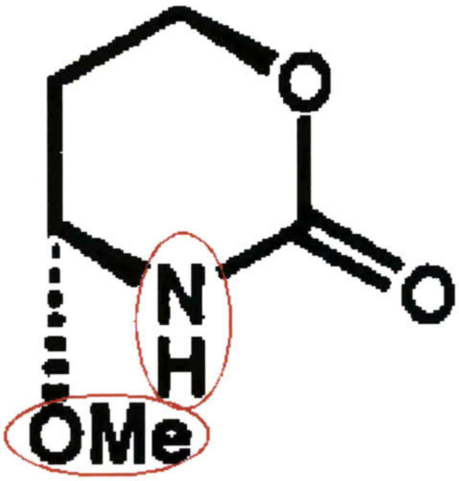


Figure 3.3: Chemical formulas written both vertically (NH) and horizontally (OMe) in the same diagram.

To measure the extent to which these shortcomings affect performance, I tested the demo versions of five OCR engines; FineReader, ReadIris, SimpleOCR [15], SoftiOCR (powered by Tesseract) [16], and PdfCompressor [17]. These engines span the spectrum from high end, expensive OCR engines to open source engines. I tested three aspects of performance: the ability to pick characters out of an image, accuracy in identifying subscripts and superscripts, and accuracy in the presence of noise. Figure 3.4 shows two of the test images that exemplify the challenges presented the OCR engines.

None of the OCR engines were able to extract the characters from the first test image reliably. As predicted, they were often able to identify the characters in the parts of the image that are arranged in a line, like "fast".

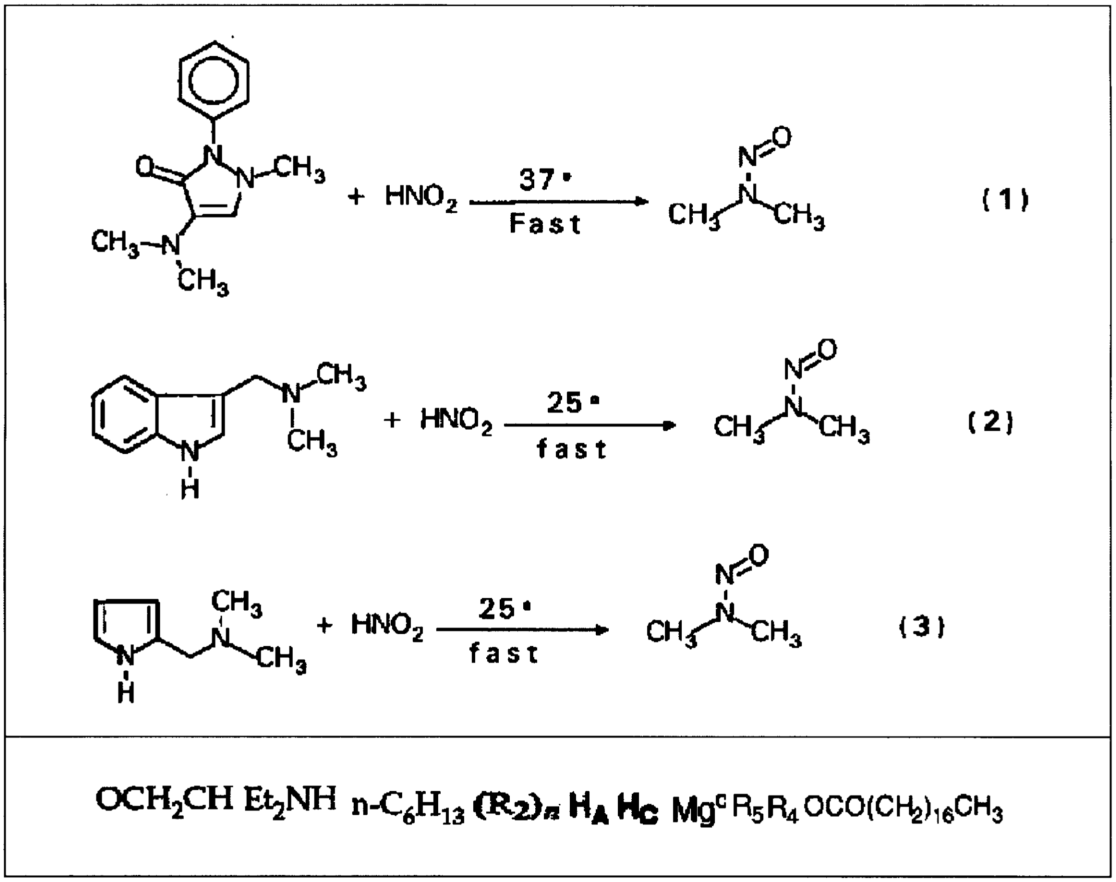


Figure 3.4: Test images. The top image contains characters that are merged with bonds as well as some fairly clean characters. The object was to test how well the OCR engines could identify characters in the image without any external help. The purpose of the bottom image was to test performance on subscripts, an integral part of molecular structure diagrams. It also contains characters that have been merged by noise. The text is arranged in a line in order to present the OCR engines input of the type they are designed to handle.

Subscripts were particularly problematic. On the second image, only FineReader and ReadIris were able to detect any subscripts at all. The others either misidentified the subscript characters or identified them correctly, but treated them as normal characters. ReadIris performed the best, correctly recognizing 37 out of the 47 symbols in the image. FineReader and

PdfCompressor were second and third best, respectively. Clearly none of these engines can be used unaided to identify text in molecular structures.

There are other problems associated with using out of the box commercial OCR engines. One of them that turned out to be of practical importance for ChemWARD is cost. Licenses for the best commercial OCR engines are quite expensive and even then in some cases it is possible to process images only on the company's servers. Another was user interface. All of these OCR engines are distributed in the form of an application with a graphical user interface that is difficult to integrate into a larger system. In light of this, for ChemWARD, there were really only two options: either choose an existing OCR engine for which the source code is available and compensate for the poor performance, or write an OCR engine specific to the molecular structure domain. While it is my opinion that the second option would be by far preferable, due to time constraints I chose the first.

ChemWARD uses the OCR engine in SimpleOCR because it turned out to be the highest performance engine for which the source code is available within the price range I was constrained to. The OCR engine itself uses a finely tuned neural network to identify characters and a dictionary file to correct mistakes. In ChemWARD, the dictionary is disabled for the reasons noted above.

I now focus on ChemWARD's OCR subsystem, shown in Figure 3.5. It operates on connected regions of pixels, produces characters, and contains an OCR engine and a Character Sense that work together to identify characters. First I describe each of the two modules, and then their interaction using an example.

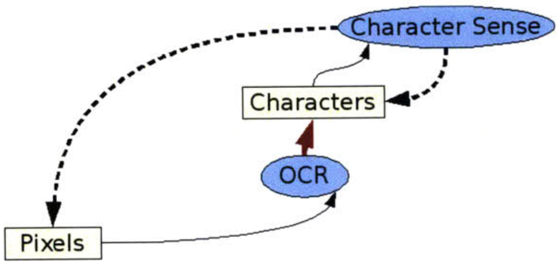


Figure 3.5: ChemWARD's OCR subsystem.

OCR engine: The test demonstrated that it is not possible to rely on an OCR engine to extract the characters from the image by itself; some external mechanism must be responsible for providing the regions of pixels that might be characters. The default strategy in ChemWARD is to break the image into connected components of pixels and apply OCR to each one. That means that in ChemWARD, each character is recognized independently; there is no concept of a "word", that is a chemical formula all at once. I found this strategy to be appropriate because in order to recognize an entire chemical formula, ChemWARD must be able to group unidentified regions of pixels together, which his difficult given how scattered the characters are. Because there is no concept of a word, a dictionary file is not useful to the OCR engine. That means that ChemWARD's OCR engine is essentially the character recognition neural network used in SimpleOCR. Given a region of pixels, the OCR engine will produce a distribution over the set of all possible characters that might be found in a molecular structure diagram. This set includes the letters A-Z, a-z, the digits 0-9, and the common symbols (,), +, -. The OCR engine also has the option of producing an empty distribution, which means that the region of pixels looks nothing like any character that the OCR engine is aware of. By producing a distribution instead of a single hypothesis, ChemWARD avoids mistakes that result from committing to a particular letter (Figure 3.6).

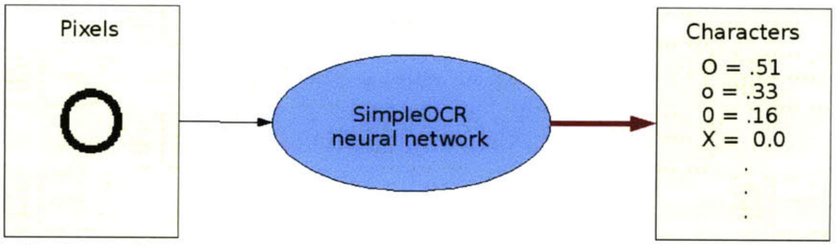


Figure 3.6: The type of output produced by the OCR sense. Without context, the pixels on the left are ambiguous; its not clear whether they represent a capital or lower case 'O' or a zero. The ambiguity is reflected in the OCR output, and will be resolved later by higher senses that have access to the required context.

Character Sense: The first pass of OCR on all the connected components of pixels will be imperfect for the following reasons:

- OCR was applied to all connected regions, including those that are actually bonds or bond groups, and it will generate false positives.
- Characters that are merged by noise will be a single connected component and the OCR sense can not perform any segmentation.
- Some characters are so noisy due to poor print or scan quality that the OCR sense will simply misinterpret them.
- Some characters look so similar that its impossible to make a decision with high confidence.

The job of the Character Sense is to refine this poor quality output as much as possible by learning as much as possible from it, correcting mistakes, and updating. In particular, the Character Sense is responsible for three tasks: rejecting false positives, segmenting merged characters, and introducing hypotheses for characters that look similar to the set of characters that the OCR engine generated. It does not try to correct misidentified characters because it does not use any contextual information; its only input is the character hypotheses themselves and the width and height of the pixel regions that the hypotheses are generated from. I now explain each of theses tasks in turn.

As mentioned, some characters are indistinguishable depending on the font. Because the OCR engine is unreliable, the Character Sense compensates by defining sets of similar characters and enforcing that if a set of hypotheses contains one of them, it must contain all of them. The Character Sense also defines special characters that correspond to bond types. In this way it is possible to specify that the pixel region might also be a bond, information that is useful to other senses. The sets of similar characters defined in ChemWARD include { 'i', 'I', 'l', '1', the single bond character }, { '-', the single bond character}, {'o', 'O', '0' }, and { 'C', 'c', 'G' }.

To accomplish the other two tasks, the Character Sense uses the widths and heights of the pixel regions that the OCR engine has identified as characters to estimate the average character size. Using estimates of the character size is consistent with the guiding principle of ChemWARD that it is easier to reject possible interpretations than to commit. In the case of character hypotheses, the Character Sense expects that connected components that are too tall are false positives, and connected components that are too wide are either also false positives, or they are characters that have been merged by noise. First I will explain how the Character Sense makes an estimate of the expected character width and height, and then I will explain exactly how it is used.

In estimating the character size, ChemWARD makes the assumption that of all the connected components that the OCR engine proposed hypotheses for, most of them are in fact characters, although they may be misidentified. False positives will be significantly larger or smaller (either in width or height) than actual characters. Each dimension, height and width, are considered independently. A simple average in a particular dimension is not a very robust estimate because the false positives can skew the estimate up or down. Similarly, the variance is sensitive to outliers. A better estimate would be the average measurement excluding outliers that are either too small or too large, but without any other information about how big a character is, its difficult to distinguish outliers from true characters. The assumption that the number of false positives is smaller than the number of properly flagged characters implies that if the measurements for the characters are plotted on an axis, the true estimate of a character size will be near the greatest density of points. Such a plot for character heights is shown in Figure 3.7. In order to locate the greatest density of points, consider each character height to be a point on the height axis. For each point, count how many points fall inside a box filter of the appropriate width centered on the point. If the assumption that there are relatively few false positives holds, and if the box filter is sized appropriately, then the points with the most neighbors in the box will be the ones that are closest together, and can be used to construct an average. It remains to define what is an "appropriate width" for a box filter. ChemWARD solves this problem by setting the width to be a fixed percentage of the average over all the proposed character measurements in the dimension (height or width).

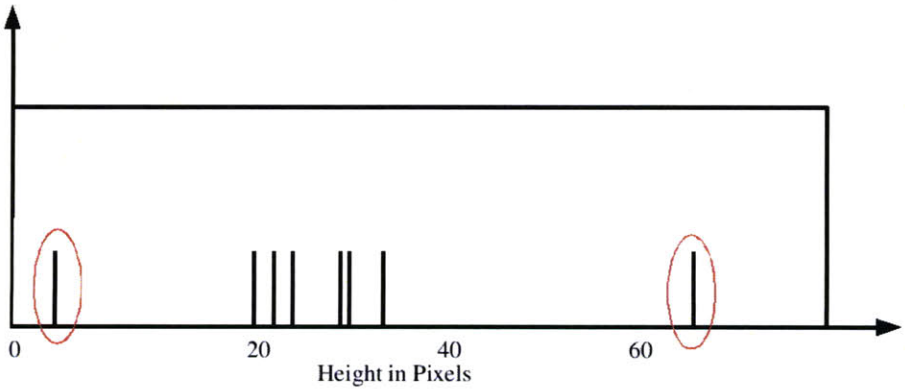


Figure 3.7: For each proposed character, plot the height. The bars circled in red represent outliers due to false positives. The true estimate for character height is the average of the bars in the middle.

ChemWARD models character height and width falling in a uniform distribution centered on the most up to date estimates. The decision to use a uniform distribution as opposed to a Gaussian is justified by the assumption that the characters typed in the underlying image were all originally typed in the same font size, so there simply are no long tails for characters to fall in.

The exact width of the uniform distribution ChemWARD assumes is percentage of each estimate, a constant parameter to the system. Pixel regions whose heights fall outside this uniform distribution are rejected as false positives. To combat merged characters, ChemWARD tries to segment pixel regions whose widths are too large to fall within the distribution. Segmentation is the most important task the Character Sense is responsible for, and is a typical example of how a high level sense can correct low level mistakes; in this case how the Character Sense can correct mistakes in the pixels that occurred during digitization of the original document.

Figure 3.8 shows a particularly insidious example of merged characters. It is a challenging image because there are multiple characters involved and one of them is a parenthesis, a relatively narrow character. ChemWARD uses the character width estimate to identify the region of pixels corresponding to the blob of merged characters as too wide to be a single character, but having approximately the right height to be text. This combination of indicators suggests the presence of merged characters, and that segmentation should be attempted. In addition to the width estimate, ChemWARD makes the assumption that the noise that merged the characters consists of only a few pixels, similar to a the font ligatures shown above, or the noise in Figure 3.8.

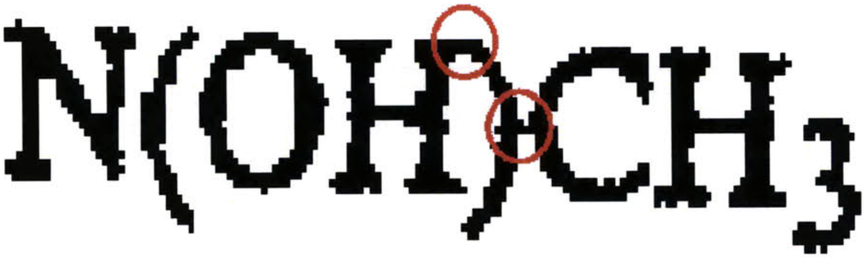


Figure 3.8: This example shows a chemical formula where noise has merged three characters together in the two places indicated by the red circles.

Under these assumptions, ChemWARD attempts to segment characters by using the width estimate to guess approximately where the break should occur, splitting the connected component of pixels by erasing vertical runs of pixels, and re-applying the OCR engine to the newly formed connected components. Figure 3.9 shows an example. The pixels to the left are the merged characters from Figure 3.8. The first step in segmentation is to use the width estimate, shown by the red arrow, to guess approximately where the merge noise is. Then ChemWARD defines a region around the estimate to search for potential ligatures, shown by the double headed arrow above the pixels. The width of the search region is a constant parameter.

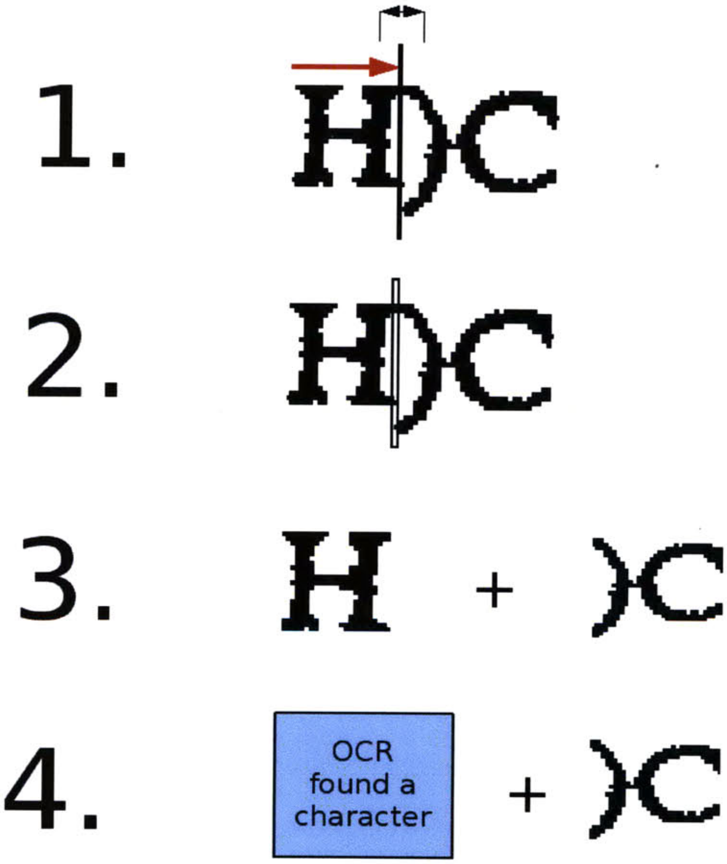


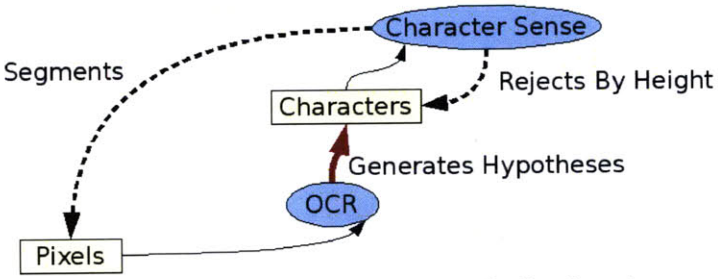
Figure 3.9: Steps to segment merged characters.

In the second step, ChemWARD sorts the vertical pixel runs by length, and hypothesizes that the noise is actually the shortest vertical run. The pixels are split at the vertical run, and the left connected component is passed to OCR in the third step. If OCR identifies the left region as a character, the pixels are permanently split at that vertical run, shown in step 4. This is how the Character Sense corrects pixel level mistakes. The segmentation procedure is repeated on the right half of the remaining pixels to separate the '(' and the 'C'. If OCR does not identify a character, ChemWARD tries the rest of the vertical runs. If OCR still does not find a character, ChemWARD declares the segmentation operation a failure.

In the case of more than two characters as in Figure 3.8, ChemWARD searches for characters by repeatedly applying the segmentation process from left to right until every connected component of pixels is identified as an appropriately sized character. If the segmentation fails, then ChemWARD rejects the original wide connected component as a false positive. One final technique that ChemWARD uses to improve segmentation performance is to sort the vertical runs of pixels by length and erase the shortest ones first, following the assumption that the noise merging the characters is small like a ligature. If ChemWARD successfully separates the merged characters by identifying subregions of the connected components that the OCR engine believes are characters, the pixel region is permanently split at the vertical runs identified during the segmentation process. By doing this, the Character Sense has just corrected a very low level mistake in the pixels; i.e., the noise that caused the characters to be merged in the first place. This correcting behavior is used in many more instances in ChemWARD that will be pointed out in future chapters.

The segmentation example also demonstrates the problem of keeping intermediate representations consistent; when the Character Sense splits the pixels, how does that affect the vectorization? If the Character Sense splits the pixels through the middle of a rectangle, clearly that rectangle is no longer valid. In fact, the correct behavior is to compute separate vectorizations for each of the newly formed pixel regions. What actually happens in ChemWARD is that when the Character Sense splits the pixels, it assigns each pixel region to a new column. The column then keeps track of the new vectorization, the new character hypotheses, and eventually all the rest of the information that becomes associated with each pixel region. That is precisely the power of the column representation.

In this chapter, I have explained the OCR problem and how it is related to the molecule recognition problem. I have highlighted some of the challenges specific to the molecular structure domain, and described ChemWARD's response to them; namely the OCR engine and the Character Sense. Figure 3.10 summarizes the interactions between the components of the OCR subsystem. It is the OCR subsystem that creates the character level representation, which we have seen consists of the distributions generated by the OCR engine and refined by the Character Sense. How to resolve those distributions into final character interpretations is the subject of the next chapter.



3.10: The OCR subsystem with labels indicating the types of interaction taking place. Unlabeled arrows mean that the sense simply examines the data.

Chapter 4

Character Aggregation

After the characters have been identified by the OCR subsystem, they must be grouped together into chemical formulas. The process of constructing and verifying the chemical formulas is carried out by the Character Aggregator.

The biggest difference between chemical formulas in a molecular structure diagram and lines of text written in a novel is that the letters (and super/subscripts) in chemical formulas are not always written left to right. Sometimes they are right to left, and sometimes top to bottom. The reason for this difference is clarity in depicting the molecular structure; sometimes it is easier to depict critical geometry by writing the characters in the chemical formula wherever there is space. Figure 4.1 illustrates this: if the NH were written left to right, it might make the hexagonal bond structure less apparent.

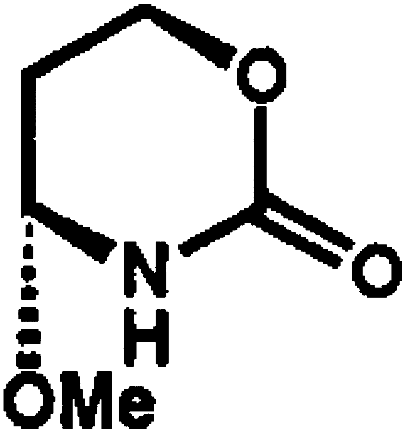


Figure 4.1: There is not enough space to write the NH left to right without obscuring the hexagonal bond structure.

Chemical formulas are also at times written right to left, but this is not a problem because the meaning of the formula remains unchanged; H_2O and OH_2 are equally valid ways of writing the chemical formula for water. In general, it is permissible to write the atoms in a chemical formula in any order. While subscripts must always appear to the right of the atomic symbol or shorthand they refer to, superscripts may appear either on the left or on the right, though it is more common for them to appear on the right. This makes using a dictionary to aid character aggregation impractical because each formula has many legitimate ways of being written.

Aside from challenges inherent in the way molecular structure diagrams are drawn, there is one additional challenge that is a result of ChemWARD's guiding principle of committing to an interpretation only at the last possible moment. At the time when the Character Aggregator processes the image, the characters have not yet been resolved into their final interpretations. Each character consists of a set of hypotheses, each of which is assigned a probability, which all together form a distribution. These distributions contain both alphanumeric characters, and special characters that indicate that the region of pixels may be a single bond that is not part of a larger bond group. Thus it is in principle possible for the Character Aggregator to mistakenly collect a single bond together with a group of real characters and designate them as a chemical formula, as in Figure 4.2. This possibility necessitates a resolution step to check the validity of the formulas that the Character Aggregator produces and at the same time resolve characters into their final interpretations. I describe the resolution step after the aggregation step.

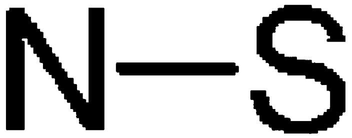


Figure 4.2: The Character Aggregator might collect all three symbols into a chemical formula if the OCR subsystem confidently misidentifies the single bond joining the 'N' and the 'S'.

As noted, using a dictionary to aid aggregation is impractical both because the dictionary would be too large and because the characters haven't been resolved yet. As a result, the Character Aggregator must rely only on the size and position of the potential characters. When processing potential characters, the Character Aggregator makes the following assumptions about how characters are written in diagrams:

- Chemical formulas are never written diagonally; it is always possible to read a chemical formula by scanning from left to right, and from top to bottom.
- When chemical formulas are written vertically, they are left justified.

The general strategy is to find characters that form a line of text. More specifically, the Character Aggregator sorts the potential characters by the minimum x coordinate of their pixel regions. It then processes each character, starting with the leftmost one, in a greedy fashion. Each leftmost character is used to define three edges of a bounding box for the formula; the left, top, and bottom edges. The right edge is used to define a margin within which the leftmost x coordinate of other characters must fall if they are to be considered part of the current formula. The margin is expressed as a percentage of the character width estimate computed by the Character Sense, and it must be wide enough to account for spacing between characters. This value is an empirically determined parameter, listed in the Appendix, along with all other system parameters. Characters whose leftmost x coordinate falls within the margin and whose highest and lowest y coordinates fall within the vertical margin defined by the bounding box are added to the current formula. The right edge of the bounding box is adjusted, and a new margin is computed. Figure 4.3 illustrates this process.

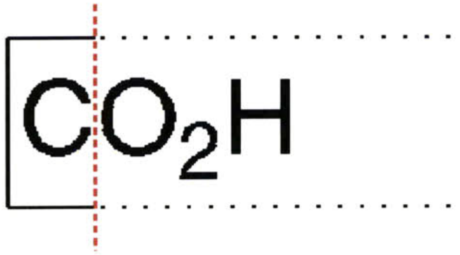


Figure 4.3: The 'C' has been identified as the leftmost character. Characters that are to be grouped with the 'C' must fall between the dashed horizontal lines. In order to be grouped with the 'C', the next character must be within a certain distance of the dashed vertical line.

To address the problem of compounds written vertically, the Character Aggregator finds already grouped lines of text whose left edges are aligned and have very little vertical distance between them. All distance thresholds used in character aggregation are listed in the Appendix.

After character aggregation is complete, each group of characters is checked for syntax errors and resolved into its final interpretation. Because it is possible for the aggregation step to mistakenly include lone bonds in a group of characters, the resolution step must also identify and reject them, splitting the group on either side of the bond. This procedure correctly separates the 'S' and the 'N' in Figure 4.2.

The first step in verifying syntax is to tokenize the group of characters by height into superscripts, characters, and subscripts. This is done by simple distance thresholds. The syntactical rules that apply to each category of token are slightly different. Subscripts are most commonly numbers, but can also be letters. Superscripts can in addition contain plus and minus signs. Standard, full height characters consist of atomic symbols and chemical shorthand. Although it was not practical to use a dictionary to check the syntax of entire chemical formulas, using a dictionary is very helpful for verifying individual tokens because the vocabulary is much smaller. When resolving characters, ChemWARD prefers interpretations that are in the dictionary. For example, when resolving the characters shown in Figure 4.4, ChemWARD will prefer the interpretation 'Cl' (Cholorine), as opposed to 'CI', even though in the sans-serif font used in the image the two possibilities are identical.

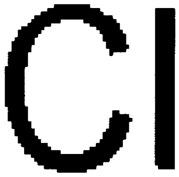


Figure 4.4: Chlorine.

The other piece of information that ChemWARD can leverage when resolving characters is the height of the character, because lower case letters are often shorter. ChemWARD can then remove upper case hypotheses from the distributions of short characters, and enforce the constraint that an upper case letter appear immediately to the left. Short characters that violate these conditions are likely to be bonds and are rejected as such. It is this type of height logic that ChemWARD uses to reject the single bond in Figure 4.2 and split the remaining characters into two formulas. After each token has been resolved, the tokens are collected together into formula objects that comprise the formula intermediate representation shown in Figure 4.5.

Resolving characters into their final interpretations is a significant commitment to a final interpretation. According to the guiding principles of ChemWARD, it should be done as late as possible. Consequently, while character aggregation could in theory take place as soon as the characters have been identified by OCR, it is actually the last task performed before the final output is produced. In particular, it happens only after all senses related to vectorization have had an opportunity to examine the data, and after large bond blocks have been converted into individual bonds. This strategy allows all the other senses to contribute their knowledge to the final interpretation.

Chapter 5

Vectorization

This chapter covers the vectorization subsystem in ChemWARD. I first present some background information on the problem and an overview of existing approaches. I then focus on how vectorization contributes to solving the molecule recognition problem, and explain some of the shortcomings of traditional algorithms in that respect. Finally I present ChemWARD's vectorization algorithm, which extracts both rectangles and circular annuli. In order to simplify the explanation of algorithm, I first describe it only for the case of rectangles and then show how it has been extended to include circular annuli.

Vectorization is the problem of approximating a bitmap image with a set of primitive geometric shapes. It is a broad problem because what it means to be a good approximation is not well defined; it depends on the application. The set of allowable geometric primitives also depends on the application; in most cases, only rectangles are used. The task of vectorization is widespread; in addition to molecular structure diagrams it is also applied to drawings of electrical schematics, mechanical diagrams and blueprints, as well as improving images aesthetically.

There are many advantages that a vector image has over a bitmap image:

Quality: A bitmap image is limited by the resolution of the pixels. When the image is magnified sufficiently, the effects of aliasing are obvious (Figure 5.1). A vector-based representation does not suffer this limitation, as it is possible to interpolate as finely as necessary on the lines that define the edges of the vectors.

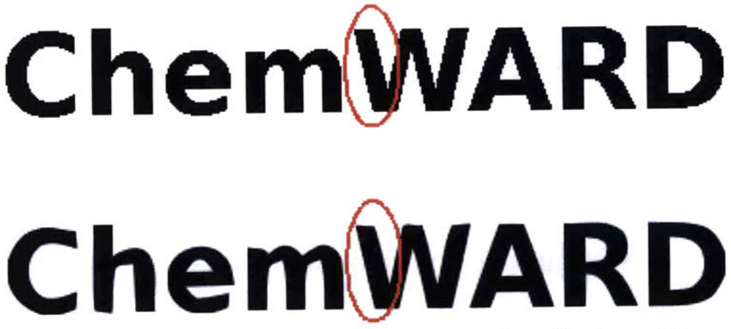


Figure 5.1: Top: bitmap image with the aliasing effects highlighted. Bottom: the same image converted to vector graphics using Vector Magic.

File Size: Storing a set of vectors generally requires less space than storing a set of pixels.

Abstraction: Because a vector representation is more abstract than a collection of pixels, certain types of manipulations become easier; for example changing the angle between two lines.

Interpretation: Vectors expose the underlying structure of an image, if there is one, much better than pixels do. For this reason, vectorization is an integral step in interpreting many types of diagrams, as mentioned earlier.

Because vectorization is so useful, many vectorization algorithms have been devised. They can be classified into three categories:

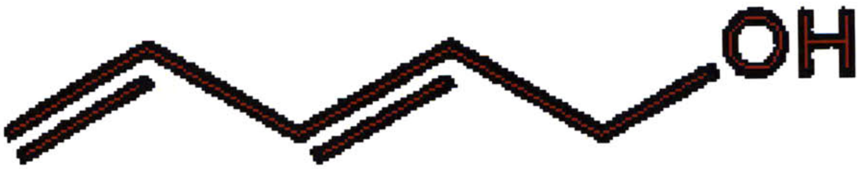


Figure 5.2: The skeleton of the image is drawn in red.

Thinning Algorithms: This type of algorithm erodes pixels from the image until only a one pixel thick skeleton of the image remains. Figure 5.2 shows the skeleton of an image. If the skeleton is determined successfully, it's possible to recover the rectangles by sampling the image perpendicular to the skeleton in order to find out how far on either side of the skeleton the black pixels extend. Algorithms that employ this technique are [18]. Today it is no longer used because the next two techniques have proven superior.

Skeleton Following Algorithms: These algorithms seek to improve upon thinning algorithms by trying to sample the skeleton directly instead of eroding pixels. To accomplish this, such algorithms start with a guess about where the skeleton is and sample in that direction, correcting the estimate as they sample. Figure 5.3 illustrates this process. Examples of this type of algorithm are [19] [20].

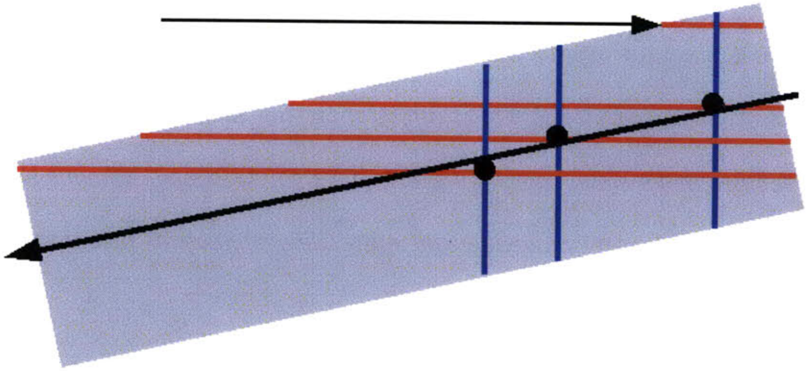


Figure 5.3: Example of a skeleton following algorithm like SPV. The algorithm first encounters pixels as it scans across the image at the tip of the top black arrow. It then scans horizontally (in red) until it reaches the end of the region, finds the midpoint of the run, scans vertically in both directions (in blue) until it locates the boundaries of the region, and considers the midpoint a sample point on the skeleton. It then uses this sample point as the starting point for a new set of horizontal and vertical scans, until it reaches the end of the region of pixels. The line that best fits the sample points is the skeleton, drawn as a heavy black arrow.

Contour Following: Instead of focusing on the skeleton of an image, contour following algorithms trace the edges of regions of pixels and produce lines or smooth curves that approximate the outline of the shape depicted by the pixels. This method is especially useful for approximating curved shapes like handwriting. Examples of contour following algorithms include [21] [22]. The vector image in Figure 5.1 was produced by a contour following algorithm.

The power of vectorization for the molecule recognition problem arises from its ability is to expose rectangles that correspond to bonds. This is particularly important when that it involves segmenting a group of bonds that have been drawn together (Figure 5.4) into constituent rectangles. Double, triple, and hash bonds are represented conceptually as line segments, but in bitmaps the constituent line segments appear to be noisy rectangles. A useful vectorization algorithm must then be able to discover rectangles very well. Vectorization is most useful for the molecule recognition problem when it produces rectangles that represent bonds, as opposed to rectangles that cover the pixels. Figure 1.3 is repeated below to underscore that point; the rectangles in (b) best cover the pixels, but the rectangle in (c) corresponds to the single bond. Vectorizations like the one in (b) should be in the molecule recognition problem because they require the extra step of reasoning to merge the two rectangles and only then conclude that the merged rectangle corresponds to the bond. In other applications, there is no notion of the correspondence between a vector and a symbol; for example in the case of beautifying images like Figure 5.1. Consequently, other vectorization algorithms may prefer more rectangles in order to get a better coverage of the pixels, and thus more faithfully represent the underlying image portrayed in the bitmap.

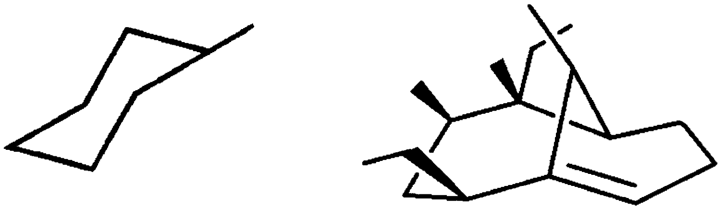


Figure 5.4: Bond groups with complicated geometry.

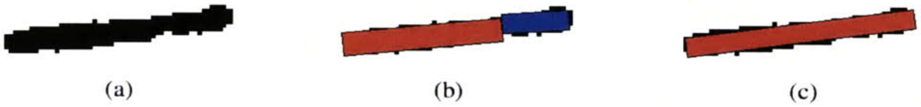


Figure 1.3: (a) is a noisy set of pixels that is really a single bond. It takes two rectangles to cover all foreground the pixels while covering as few of the background pixels as possible, as shown in (b). Because (a) is really a single bond, the desired outcome is to approximate the region with a single rectangle like (c), even if the pixel coverage is not as good as in (b).

I chose not to use a contour following algorithm because those types of algorithms generate outlines of the pixel regions instead of rectangles. Because molecular structure diagrams are known to be composed of noisy rectangles, using a contour following algorithm to produce outlines and then converting those outlines into rectangles is adding extra work, complexity, and opportunity for error. I also decided against contour following because the most noise occurs on the edges of regions of pixels – precisely where contour following algorithms operate. A noise robust vectorization algorithm should focus on the interior of pixel regions to produce rectangles.

I also decided against using a traditional skeleton following algorithm due to the breakages that occur in Figure 1.3. Such breakages could occur for two reasons: either the algorithm is tuned to maximize pixel coverage without sufficient restriction on the number of rectangles it generates, or noise corrupts the image so much that the skeleton that the algorithm is trying to sample is altered as in Figure 5.5. In order to overcome these problems, ChemWARD fits rectangles to regions of the image, and instead of measuring performance in terms of how the pixels are covered, ChemWARD measures performance in terms of the quality (according to a scoring function defined later) of the vectors extracted. Because bonds are usually made of long, thin rectangles, a good quality vector is also a long, thin rectangle.

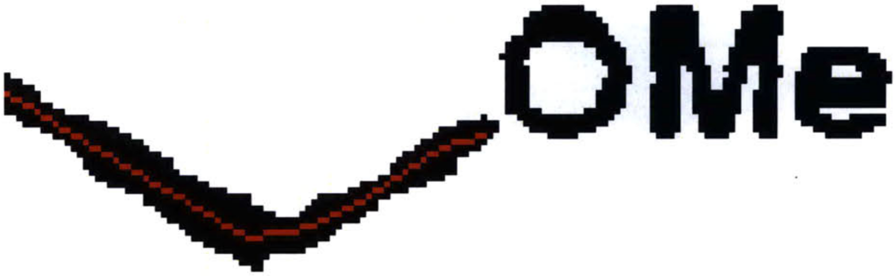


Figure 5.5: Noise has corrupted this image so much that the skeleton, drawn in red, no longer consists of two line segments; it has become a bit wavy. Skeleton sampling algorithms will approximate a wavy skeleton with several rectangles, causing the breakages shown in Figure 1.3.

In ChemWARD, the task of extracting a rectangle from a region of pixels is broken down into two subtasks:

- Identifying a part of the image where a long rectangle is likely to fit.
- Fitting a rectangle to that region.

In order to formulate these two tasks more precisely, let us consider the outline of a region of pixels to be a polygon. Each edge of the polygon is one pixel long. Even though it is sometimes possible to aggregate many of these small edges into one large one, it is more useful to consider them separately, as we shall see. Figure 5.6 shows a region of pixels and its corresponding outline.

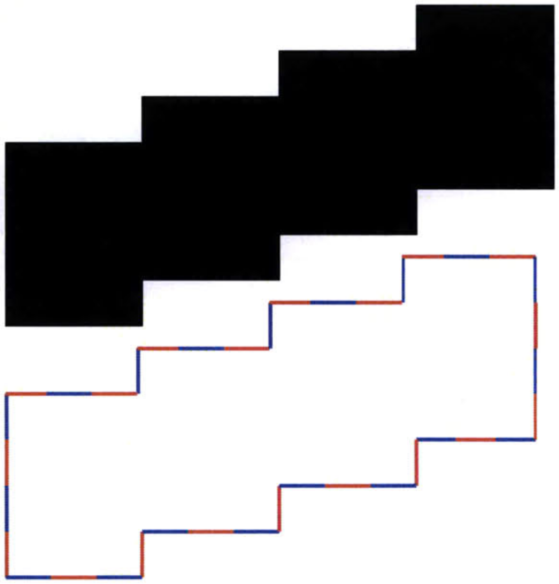


Figure 5.6: A region of pixels and the corresponding polygon defined by the pixel boundaries. Each edge in the polygon is one pixel wide.

In order to identify a region of pixels where a long, thin rectangle is likely to fit, let's solve a slightly different problem. Let's try to find the largest rectangle that fits inside the polygon. If the underlying pixels represent a bond in the molecular structure diagram, the largest rectangle in the polygon must be long and thin because bonds are long and thin. This is why finding the largest rectangle that fits in the polygon is in fact a good indicator of where a long, thin rectangle is likely to fit inside the region of pixels. If the underlying pixels represent a letter, then the result of the vectorization algorithm is irrelevant because the OCR subsystem handles character detection and will discard the vectorization anyway. Figure 5.7 shows the largest rectangle for the polygon in Figure 5.6. For the moment, we ignore aliasing.

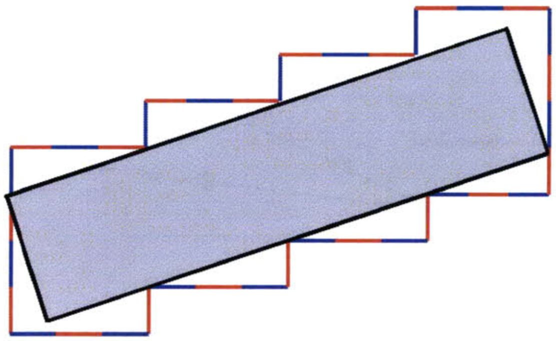


Figure 5.7: The largest rectangle that fits inside of the polygon.

Notice that wherever a large rectangle fits inside the polygon, a long line segment will also fit inside the polygon; namely the diagonal of that rectangle. If it is possible to find the longest line segment that fit inside the polygon, it might then be possible to find where the largest rectangle fits. Furthermore, it is not necessary to find an exact analytics solution for the longest line segment. When we go back to fitting a long thin rectangle into a region of pixels instead of a large rectangle in a polygon, the precision of the solution will be limited by the resolution of the pixels. In particular, the final rectangle will have verticies with integral coordinates because we are working in the discrete space of pixels. We thus have a certain amount of slack when finding the longest line segment.

Because we have this slack, it is possible to recast the problem to that of finding, for each pixel in the interior of the polygon, the longest line segment that both passes through that pixel and is contained entirely in the polygon. Figure 5.8 shows these line segments for some pixels in our example polygon. The longest line segment that fits in the polygon will approximately equal the longest line segment that passes through one of the interior pixels as long as the region of pixels is more than one pixel thick.

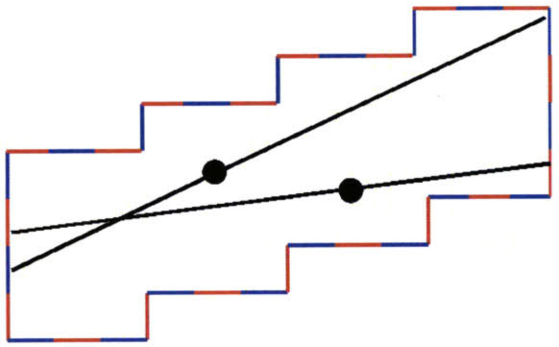


Figure 5.8: The black dots depict interior pixels. The line segments through them depict the longest line segments that both pass through the pixels and are contained in the polygon.

Due to the slack mentioned above, for any interior pixel *p,* its not necessary to find precisely the longest line segment that passes through it. Because every edge is one pixel long, it is sufficient to identify the two edges of the polygon that contain the endpoints of the longest line segment. By identifying these two edges, we have identified the endpoints of the longest line segment at the maximum precision allowed in the space of pixels. The rasterized endpoints of the line segment will fall on the pixels that correspond to the two edges that contain the endpoints. To identify the edges where the line segment endpoints fall, define a circle with a unit pixel radius centered on the interior pixel *p,* and project each edge of the polygon onto the circle, such that each edge defines an interval of angles on the circle, some of which are shown in Figure 5.9. Notice that in order for this simplification to be valid, it is absolutely critical that each edge of the polygon be only one pixel long, even if the more natural way to represent the polygon would be to merge several adjacent single pixel long edges into one edge that is multiple pixels long.

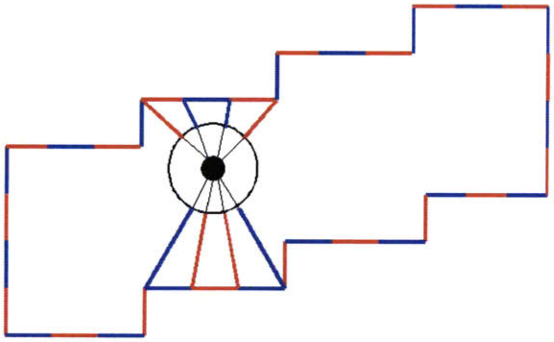


Figure 5.9: Depicted are some of the edges of the polygon projected onto a unit circle centered at an interior pixel. Each edge defines an interval of angles on the circle.

It is possible for intervals defined by different edges to overlap (Figure 5.10). In the case of overlap, we give precedence to the edge whose midpoint is closest to the interior pixel, because any line segment passing through the pixel at an angle in the overlapping interval must pass through the closer edge first. The edge whose midpoint is closer to the interior pixel is also closer over its entire length because edges in the polygon never cross, so when determining which edge is closer, it is only necessary to check which midpoint is closer. Because the polygon is closed and the pixel is in the interior, projecting every edge onto the circle must cover the entire circle. By projecting every edge onto the circle, we are guaranteed to partition the entire circle into intervals of angles. By always preferring the closest edge, we guarantee that each edges associated with an interval is the first edge that a line segment traveling in a direction in that interval will encounter.

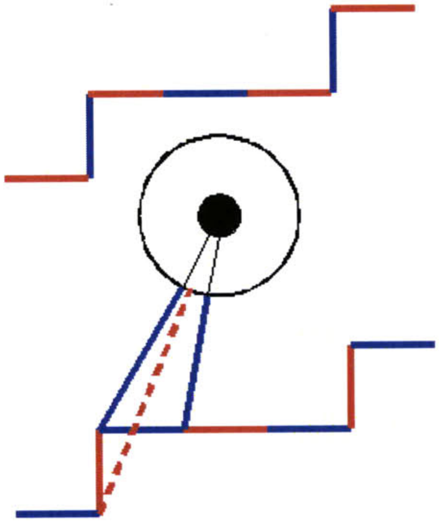


Figure 5.10: The red edge projects onto an interval that overlaps the blue edge's interval. The blue edge is preferred because it closer to center of the circle.

To find the edges that the longest line segment through the pixel must cross, traverse the circle and for each unique pair of angle intervals (a, $a + \Pi$), compute the distance from the pixel to the edge associated with a and the distance to the edge associated with $a + \Pi$. The maximum distance over the circle is approximately the length of the longest line segment that passes through the pixel and is contained in the polygon. The edges associated with the angle intervals in the maximum distance pair are the edges that the longest line segment through the pixel must cross. Figure 5.11 shows the longest line segment for our example polygon and its corresponding angle intervals. The pixels that correspond to these edges then define the longest line segment through the pixel when the line segment is rasterized. In order to find the longest line segment that is contained in the polygon, find the longest line segment that passes through each of the interior pixels, and choose the longest one[1].

[1] I subsequently discovered that Hall-Holt et. al. published an algorithm that solves the this problem precisely [23], but have not used it here because I discovered their algorithm only shortly before finishing this thesis.

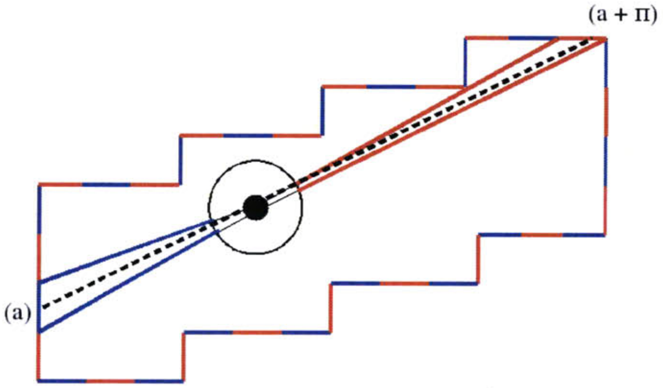


Figure 5.11: The longest line segment must pass through the edges associated with some angle interval (a) and its corresponding (a + π). To find the longest line segment, walk along the circle, checking the length of the line segment at each opposite pair of intervals.

Having found the longest line segment that fits in the polygon, the next task is to determine how it should be used to fit a large rectangle to the surrounding neighborhood of pixels. ChemWARD accomplishes this using a discretized version of gradient ascent. The object of the search is to find the largest rectangle that covers the most foreground pixels while simultaneously covering the fewest background pixels. It is important that rectangles not be forbidden from covering background pixels (and vice versa) because noise may make the edges of the pixel regions ragged and the vectorization algorithm should be robust to that. As an initial best fit, the line segment found previously is converted into a one pixel wide rectangle. The search proceeds by exploring a set of similar rectangles for possible better fits. These rectangles are generated by expanding, shrinking, or skewing the edges of the current best fit. Each of the new rectangles is rasterized and a scoring function is applied. The scoring function used in ChemWARD is:

λ_1 * dark pixels on the boundary +

λ_2 * dark pixels in the interior +

λ_3 * light pixels on the boundary +

λ_4 * light pixels in the interior +

λ_5 * dark pixels on the boundary that belong to another rectangle +

λ_6 * dark pixels in the interior that belong to another rectangle +

λ_7 * light pixels on the boundary that belong to another rectangle +

λ_8 * light pixels in the interior that belong to another rectangle +

λ_9 * dark pixels total +

λ_{10} * light pixels total +

λ_{11} * the length of the major axis * the dot product of the major and minor axes +

λ_{12} * the length of the major axis

λ_{13} * the length of the minor axis

The weights λ are listed in the Appendix. I determined the current weights by experimentation. A more rigorous method is preferable, and I discuss ways to improve the weights in chapter 8. The vectorization algorithm updates the best fit rectangle as described above until no more improvement is possible. The resulting rectangle is declared to best fit the region.

After the best fit rectangle has been found for one region of pixels, the procedure must be repeated for the rest of the black pixels in the image until all the pixels have been claimed by some rectangle, or have been discarded as noise. This requires an update step to account for the fact that long line segments running through rectangles that have already been identified are not as as valuable as long line segments that run through unclaimed pixels. Every pixel inside or on the boundary of a best fit rectangle is excluded from the set of pixels from which line segments are computed. A pixel may also be excluded if more than half of its neighbors are background pixels or pixels that have been already been claimed. In addition, for the remaining pixels under consideration, ChemWARD discounts the length of the longest line segment through the pixel by the distance that it travels through any already identified rectangle (Figure 5.12).

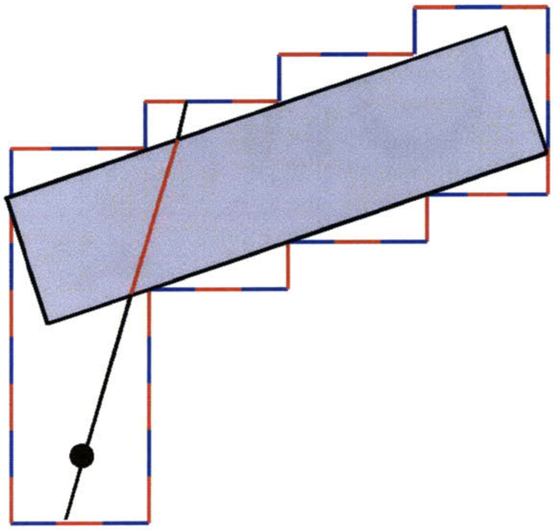


Figure 5.12: The gray rectangle has already been identified as the largest rectangle that fits in the upper region of the polygon. The black dot is an unclaimed interior pixel, with the longest line segment that passes through it but remains in the polygon drawn in black and red. Because the red part of the line segment passes through an already identified rectangle, it does not count towards the length of the line segment.

ChemWARD repeats the procedure of choosing the longest updated line segment and extracting the best fit rectangle until there are no more unclaimed pixels.

As mentioned earlier, ChemWARD extracts not only rectangles but also circular annuli. Extracting circular annuli turns out to be a straightforward extension of the vectorization algorithm just presented, but first I explain why circular annuli are useful in the molecule recognition problem.

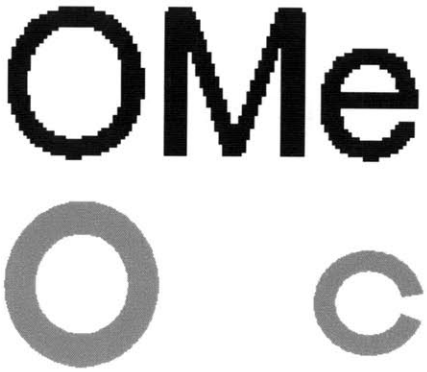


Figure 5.13: Characters often contain circular annuli, highlighted in gray.

In molecular structure diagrams, characters are often merged with bonds. Such a merged structure is unrecognizable to the OCR engine, so some other measures must be taken to segment the bond and the character. Many characters contain arcs (i.e., 'C', 'O', etc.), which when printed appear as as circular annuli of pixels (Figure 5.13). Bonds, on the other hand, are almost always drawn as straight lines. The presence of an annulus in a region of pixels is a strong indication that the region of pixels actually is a character. This technique was used successfully to separate the characters from the bonds in Figure 5.14.

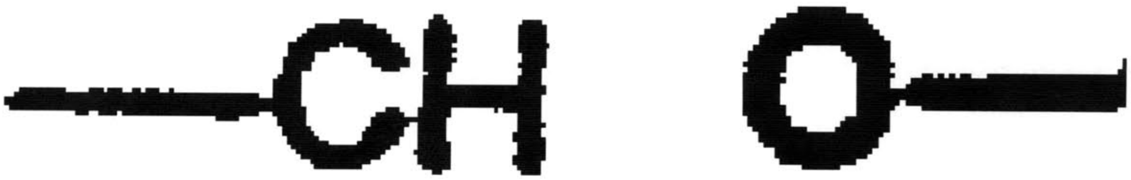


Figure 5.14: Merged bonds and characters that were separated by noticing the presence of an annulus.

Analogous to the case with rectangles, in order to detect annuli the vectorization algorithm solves two subproblems:

- Locating a region of pixels likely to contain a large annulus.
- Fitting an annulus to the region.

Because circular annuli can also be rasterized, it is possible to reuse the discrete gradient ascent technique that worked for rectangles to solve the second subproblem. But, as is usual with gradient ascent, the quality of the result can depend heavily on the quality of the initial guess. Similar to the rectangular case, it is important to notice that where ever a large annulus fits inside a region of pixels, a large circular arc must also fit, namely the outer arc that forms the outline of the annulus. The initial guess for an annulus must then be based on a circular arc. The natural question to ask is whether there is some algorithm analogous to the longest line segment algorithm used above, but which produces the longest arc instead of the longest line. To my knowledge, there is no fast way to solve this problem. But, it is a computationally simple task to fit a circular arc to a collection of points. If its easy to fit a circular arc, then its possible to walk along the boundary of each region of pixels and try to fit an arc at each step. When the error between the pixels and the best fit arc exceeds a threshold, remove the earlier pixels from the set being considered until the error drops to below the threshold. The set of arcs with sufficiently low error is the set of initial guesses for the gradient ascent routine. The longest arc is converted into a one pixel thick annulus, and new annuli are generated by increasing and decreasing the extent and radii of the current best fit. The scoring function for annuli is:

λ_1 * dark pixels on the boundary +

λ_2 * dark pixels in the interior +

λ_3 * light pixels on the boundary +

λ_4 * light pixels in the interior +

λ_5 * dark pixels on the boundary that belong to another rectangle +

λ_6 * dark pixels in the interior that belong to another rectangle +

λ_7 * light pixels on the boundary that belong to another rectangle +

λ_8 * light pixels in the interior that belong to another rectangle +

λ_9 * dark pixels total +

λ_{10} * light pixels total +

λ_{11} * the thickness of the annulus

Again, the λ are listed in the Appendix. As in the rectangular case, the weights were determined by trial and error. With that, two tasks remain to be solved; fitting an arc to a set of points, and devising an update step analogous to the rectangular case.

To fit an arc to a set of points, define a third dimension $z = x^2 + y^2$, where x and y are the coordinates of the point in the plane. In this space, points that lie on a perfect circle in two dimensions lie on a plane in three dimensions. To deal with the fact that the points are noisy and do not lie on a perfect circle, the eigenvector associated with the smallest eigenvalue of the covariance matrix of the points defines the normal to the best fit plane. This method of arc fitting is also described in [24].

The update step is identical to the rectangular case; for each remaining candidate arc, the distance that it passes through an already identified annulus or rectangle is subtracted from its actual length in order to compute its priority. Each candidate line segment is also updated by subtracting the distance that it passes through annuli.

The extension of ChemWARD's vectorization algorithm is thus straightforward. The only novel step is how the algorithm decides between fitting the longest rectangle to a region or the longest annulus. The answer is that ChemWARD's vectorization algorithm is biased towards fitting rectangles where possible, but fits an annulus if the length of the candidate arc is a constant parameter (listed in the Appendix) longer than the length of the candidate line segment.

In this chapter, I presented ChemWARD's novel vectorization algorithm. While the raw vectorization is generally high quality, it does need to be refined. The refinement step is the subject of the next chapter.

Chapter 6

Vectorization Refinement

The vectorization refinement subsystem is the largest and most complex part of ChemWARD. It is responsible for converting the raw vectorization into the bonds that form the skeleton, or the major bond groups, of the molecular structure. The chemical formulas that are produced by the character aggregation subsystem described in chapter 4 are combined with this skeleton to produce the final chemical graph output. This subsystem is responsible for extracting most of the topological information from the diagram. The simplest way to present this subsystem is first to provide an overview of the tasks it must accomplish, then describe each component in the subsystem from the bottom up. There are a number of empirically determined parameters used in the vectorization subsystem. Because these parameters were empirically determined, I refer to them here only in general terms. The actual values are listed in the Appendix.

The goal of the vectorization subsystem is to identify bonds. This includes single, double, triple, and perhaps even quadruple bonds. Single bonds are further classified as hash bonds, standard bonds, or wedge bonds. Collecting rectangles into logical groups must be a central task in producing double, triple, quadruple, hash, and split bonds. Another central task is defining bond topology. Bonds that share a chemical formula or an implicit carbon atom are said to be adjacent. Because hash bonds (and sometimes other types of bonds) don't share pixels with the bonds they are adjacent to (Figure 6.1), simple pixel overlap is not a reliable way of inferring topology, so more sophisticated methods must be employed.

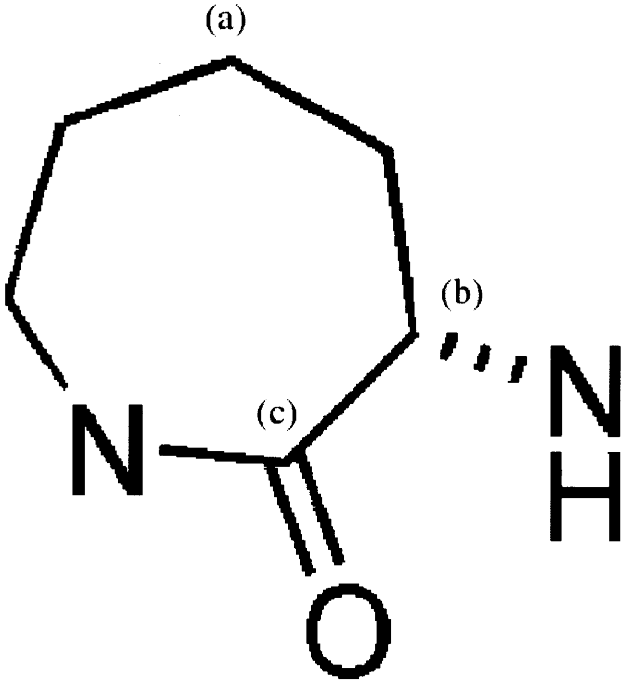


Figure 6.1: Topological challenges. In (a), the bonds overlap. In (b), the hash bond does not overlap the two adjacent standard single bonds. In (c), all four standard bonds share a single carbon atom, but they do not all overlap at the same point.

With this in mind, the specific recognition tasks of the vectorization refinement subsystem, from low level to high level, are:

Vector Correction: Removing spurious or erroneous vectors either by deleting them or by merging them. These vectors arise from imperfections in the vectorization process.

Vector Topology: Inferring which vectors should be considered adjacent, without considering them to be bonds yet. Because the vectorization process is imperfect, it is possible that rectangles that should overlap do not, even in cases like Figure 6.1 (a). In addition, structures such as Figure 6.1 (c) complicate this task.

Hash Aggregation: Collecting vectors into groups that are hash bonds. A hash bond may have a rectangular or a triangular outline (Figure 6.2). Aside from this, noise often causes some of the rectangles that are part of a wedge bond to merge with other nearby bonds or characters. It is therefore challenging both to identify which rectangles are candidates to be part of a hash bond, and to determine if a set of rectangles actually is a wedge bond.

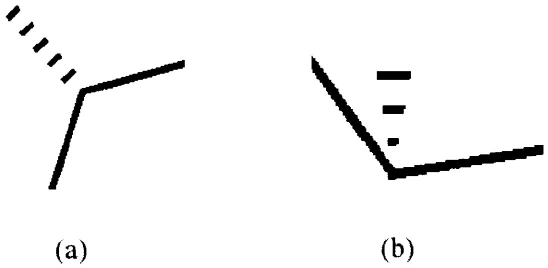


Figure 6.2: Hash bonds with both rectangular (a) and triangular (b) outlines.

Stereochemistry Identification: Determining which single bonds are standard and which are wedge bonds is difficult because a bond that extends out of the page towards the viewer may not have a wedge shape, but should still be classified as a wedge bond (Figure 6.3). Often, wedge bonds are difficult to distinguish from standard bonds when the diagram is drawn with thick standard single bonds.

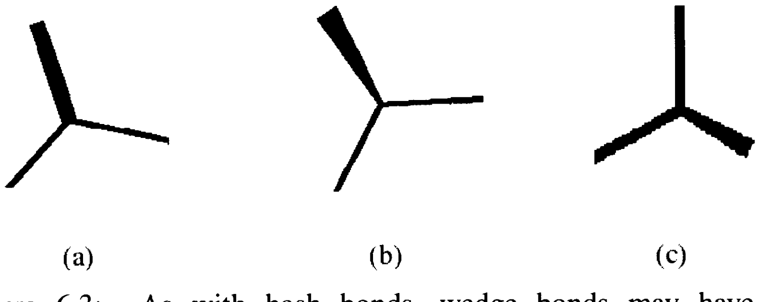


Figure 6.3: As with hash bonds, wedge bonds may have either a rectangular (a) or triangular (b) outline. When standard single bonds are thick, its hard to identify wedge bonds. The lower right bond in (c) is a wedge bond.

Lone Bonds: It is often the case that a bond connects two chemical formulas and is not attached to the skeleton of the molecular structure diagram. It's impossible to distinguish these bonds from minus charges, Iodine atoms, or characters such as lower case 'l' that might be part of a chemical formula because the system has not constructed the contextual information that makes it so obvious to a human viewer. These entities should be identified, but not forced into either a bond or character interpretation at this level. Figure 6.4 shows some examples of entities that are at this point indistinguishable from a lone single bond. Notice that size is not a good indicator to distinguish between bonds and other chemical entities.

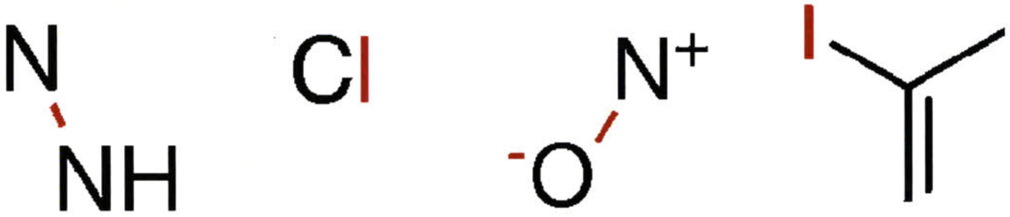


Figure 6.4: Examples of lone bonds and other chemical entities that they might be confused with.

Double, Triple, Quadruple, and Split Bonds: Like hash aggregation, the separate rectangles that comprise these types of bonds must be grouped together.

The vectorization subsystem must therefore accomplish a number of challenging tasks. To do this the vectorization subsystem employs several senses and intermediate representations. Figure 6.5 shows the architecture of this subsystem. I will proceed from the bottom up.

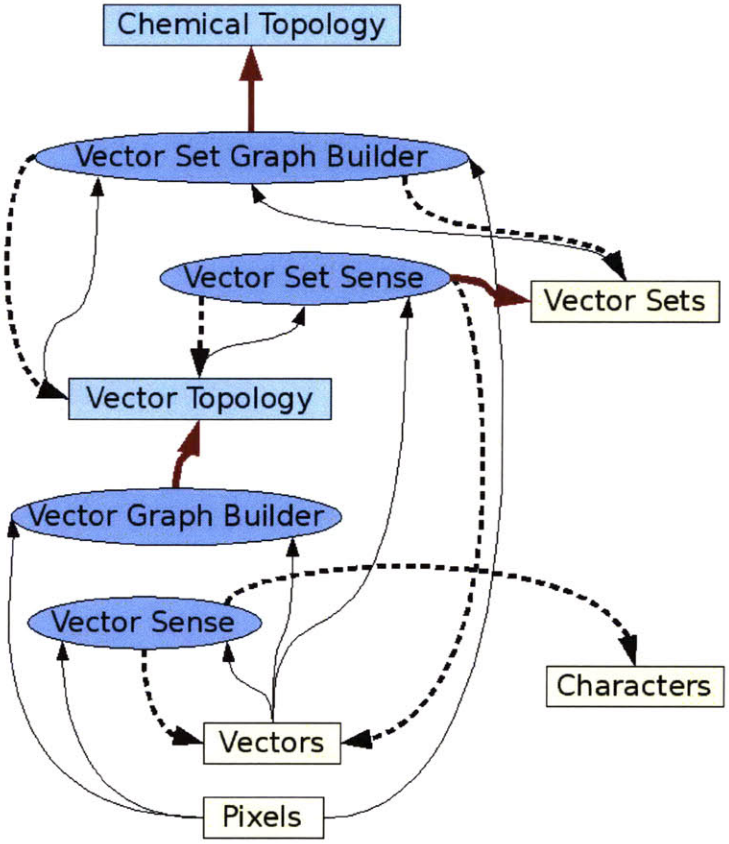


Figure 6.5: Vectorization refinement subsystem architecture. Ellipses are senses, rectangles are intermediate representations. Heavy red lines indicate that a sense produces an intermediate representation. Dashed lines indicate that a sense modifies an intermediate representation, and thin lines indicate that a sense examines the data in an intermediate representation.

The vectorization refinement subsystem is so named because it operates primarily on vectors. The first sense to examine the raw vectorization output is the Vector Sense. The Vector Sense attempts to refine vectors by consolidation. In particular, it employs several heuristics that merge vectors. These are as follows:

General Overlap: Pairs of rectangles that overlap for more than about half of their areas are merged (Figure 6.6). The merging procedure replaces the pair of rectangles with the smallest rectangle that covers all the area that the pair covered. This rectangle is obtained by taking the union of the two sets of pixels that comprise the areas of the rectangles, and then taking the convex hull. From the convex hull, it is possible to compute the minimum area bounding rectangle by using a theorem that states that two sides of the minimum area bounding rectangle must be parallel to one side of the polygon that forms the convex hull [25].

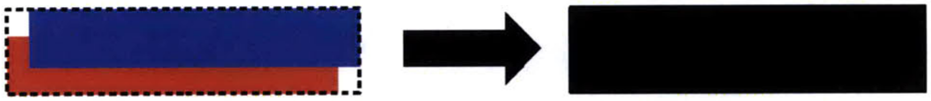


Figure 6.6: Two overlapping rectangles are merged.

Border Overlap: Again for rectangles, pairs whose border areas overlap for a certain percentage are merged. Again the merge occurs by taking the minimum area bounding rectangle for the union of areas of the pair of rectangles.

Recover Lost Pixels: If for a region of pixels there is a rectangle in its vectorization that covers most of the area, replace the vectorization with the minimum area bounding rectangle of the entire region. This heuristic is especially helpful under noisy conditions where the best fit rectangle does not cover the ragged edges of the region.

Nearly Parallel Rectangles: Pairs of rectangles that are nearly parallel and in close proximity are replaced with the minimum bounding rectangle. This heuristic is for approximating wedge bonds with a single rectangle. Because the vectorization algorithm is restricted to fit rectangles as opposed to triangles, wedge-like shapes tend to be approximated by two nearly parallel rectangles. This is illustrated in Figure 6.7.

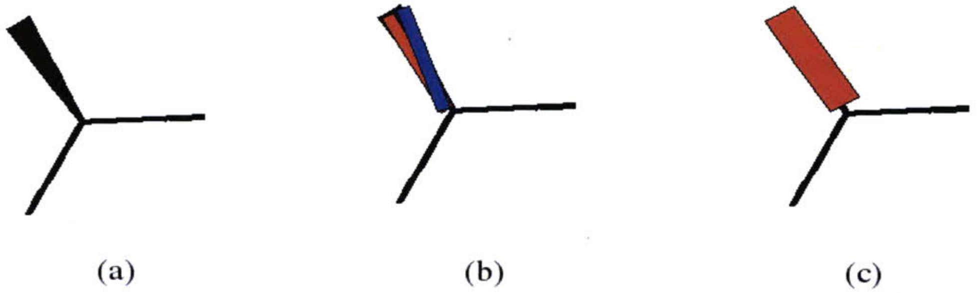


Figure 6.7: Merging nearly parallel rectangles. The original pixels are in (a). The crude vectorization is in (b); the wedge bond gets approximated by two rectangles that are close together and nearly parallel. The desired output is shown in (c); a single rectangle covering the entire wedge bond. The Vector Sense executes a heuristic to replace pairs of rectangles like the one in (b) with single rectangles like the one in (c).

Nearly Concentric Annuli: Because in some fonts characters are comprised of elliptical annuli instead of circular annuli, it is possible for the vectorization algorithm to approximate a single annulus as two nearly concentric ones. This heuristic merges the two by replacing the pair with the average center, the maximum extent, the minimum inner radius, and the maximum outer radius.

After this crude consolidation, the Vector Sense executes one more task to aid the OCR and Character Aggregator senses. If a region of pixels consists of a single large rectangle, the Vector Sense injects into the distribution of character hypotheses a special character indicating the possibility that the region of pixels is actually a lone single bond. By injecting this character, the Vector Sense flags the region for further consideration by higher level senses.

Once the Vector Sense is finished, the vectorization is clean enough to begin to construct the topology of the vectors. Ultimately this vector topology will be the basis for the bond topology. First I will define the terms relevant to vector topology, and then describe how the Vector Graph Builder constructs the vector graph.

Vectors are said to be adjacent when they meet either of two criteria:

- Criterion 1: They overlap.

- Criterion 2: A pair of rectangles does not overlap, but the line segment defined by an endpoint *e* of the medial axis of one rectangle and its projection *p* onto the medial axis of the other rectangle passes entirely through the black region of pixels that both rectangles are associated with. Figure 6.8 illustrates this case.

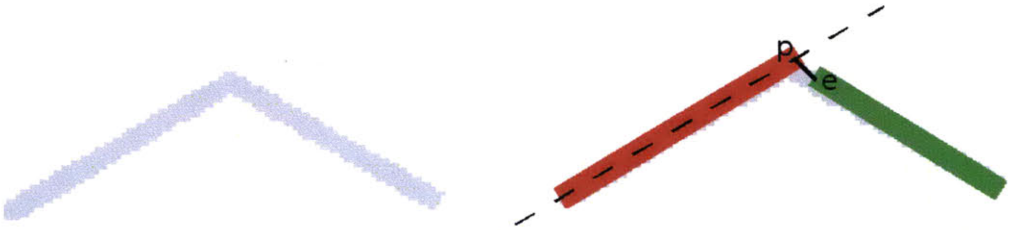


Figure 6.8: On the left are the original pixels in gray. On the right, the vectorization algorithm computed the red and green rectangles to approximate the gray region. Even though they do not overlap, the red and green rectangles are considered adjacent because the line segment between *e* and *p* passes entirely through the gray region. The dashed line is the medial axis of the red rectangle.

Pairs of adjacent vectors are neighbors in the vector graph. Each set of adjacent rectangles has a geometric point associated with it that I call a junction. All rectangles associated with a given junction are adjacent to each other, forming a clique in the graph. When constructing junctions, if the rectangles are adjacent according to the first criterion, the junction is the geometric center of the pixels shared by all the rectangles at the junction. If the rectangles are adjacent according to the second criterion, the junction is the projection of the endpoint *e* onto the medial axis of the other rectangle. Refinement heuristics can adjust the position of a junction after it has been constructed.. This means that a junction does not have a strictly defined geometric relationship with the vectors in the clique. Junctions will later be converted into implicit carbon atoms at a much higher level.

The Vector Graph Builder is responsible for constructing the vector graph that consists of adjacent vectors and associated junctions, and then refining the initial graph. It constructs the graph in five steps. The first step is to check for simple pairwise overlap that satisfies Criterion

1. It is important to note that a pair of vectors may be associated with multiple junctions in the case of annuli (Figure 6.9). When an annulus overlaps another vector in more than one place, a junction is generated for every connected region of overlapping pixels.

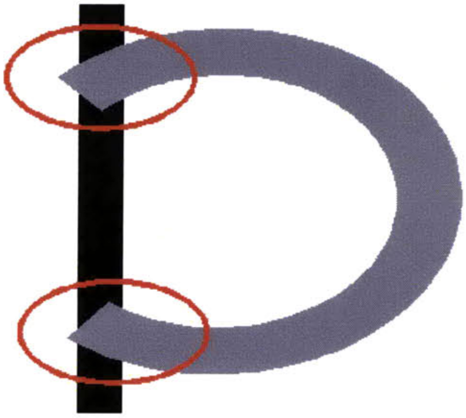


Figure 6.9: The black rectangle overlaps the gray annulus. The disjoint regions of overlap are circled in red. Because the regions from two connected regions of pixels, two junctions are generated.

The second step is to check for pairwise adjacency according to Criterion 2. This step is applied only to rectangles because annuli do not have well defined medial axes (they have medial arcs). In addition to meeting the conditions in Criterion 2, the line segment connecting the medial axis endpoint of one rectangle to the major axis of the other must be substantially shorter than the major axes of both of the rectangles. This final check prevents spurious junctions from being generated between configurations of bonds shown in Figure 6.10. It prevents a junction from being constructed between the red rectangle and the top green horizontal rectangle, even though the extensions of their major axes meet Criterion 2.

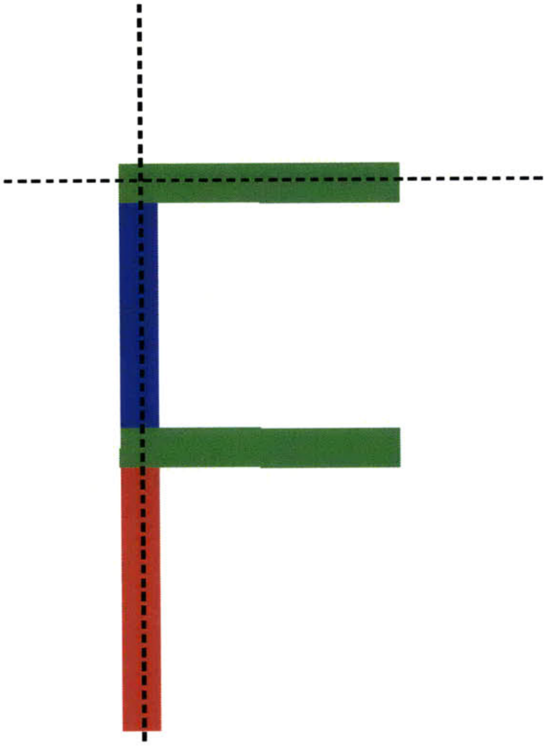


Figure 6.10: A junction should not be constructed between the red rectangle and the top green rectangle.

By this point, the pairwise topology has been established. The goal, however, is not pairwise topology but rather to identify collections of rectangles that form cliques; these cliques suggest the presence of an implicit carbon atom. As mentioned above, finding the bonds and implicit carbons is the ultimate goal of the vectorization subsystem.

The third step in the Vector Graph Builder is therefore a refinement heuristic that is applied to the junctions produced by the previous two steps. Figure 6.11 depicts a typical situation after the first two steps are complete. Notice that in the image on the left side there are three separate junctions; one for each pair of vectors. This is the natural result because each

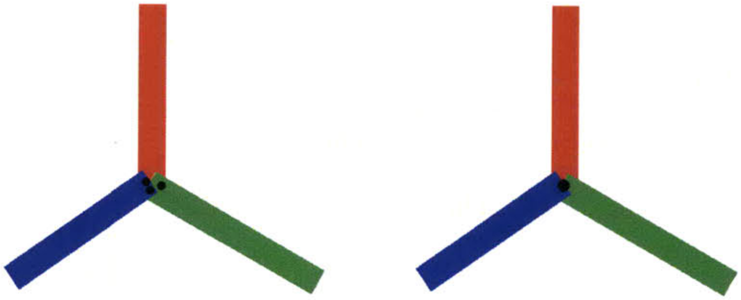


Figure 6.11: On the left are the junctions computed from the overlapping area of each pair of vectors. On the right is the desired result: all three vectors share a single junction.

junction is the geometric center of the pairwise overlap. Because the set of overlapping pixels is different in each case, the geometric center and hence the junction will be different in each case as well. The desired output is shown in the image on the right; since all three vectors overlap, they should form a clique and share a single junction. In order to consolidate junctions that logically should belong together in this way, the Vector Graph Builder locates all the 3-cycles in the graph, and if the junctions associated with each pair are close enough together, replaces the three separate junctions with their geometric center.

The fourth step in refining the vectorization involves splitting rectangles that the vectorizer (not the Vector Sense) mistakenly merged. This situation can occur when adjacent bonds have a 180 degree bond angle, as shown in Figure 6.12. The raw vectorization shown in Figure 6.12 (b) is the correct vectorization because the two rectangles do indeed approximate the pixels. Nevertheless, to a human viewer it is clear that there are four bonds instead of two. In order to split the rectangles and produce the vectorization shown in Figure 6.12 (c), the Vector Graph Builder leverages the information that bonds in molecular structure diagrams have junctions near their end, not in their middle regions. The Vector Graph Builder examines all rectangles and splits the ones that have junctions in places other than their ends, until all the remaining rectangles only have junctions near their ends. Also note that the four vectors in Figure 6.12 (c) form a clique because they all share a single junction (not shown); the splitting procedure has exposed a carbon atom for other senses to notice.

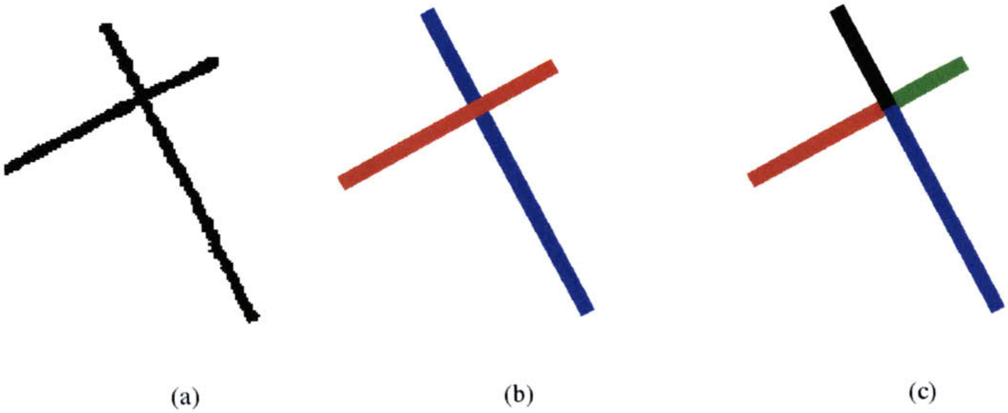


Figure 6.12: Two pairs of bonds with zero bond angle. In (a), the original pixels are shown. (b) is the crude vectorization; it correctly approximates the region with two rectangles because it has no other contextual information. In (c), the original two rectangles are split into four rectangles, each of which corresponds to one of the four bonds.

The final step in building the vector graph takes advantage of a piece of chemical knowledge: In most cases, carbon atoms support only four bonds. A clique of vectors with more than four members is a good indication that there is a vectorization mistake because such cliques generally do not arise naturally. Such cliques might occur, however, when a bond rectangle extends into the area covered by a character, because characters often are vectorized by many small rectangles and annuli (Figure 6.13).

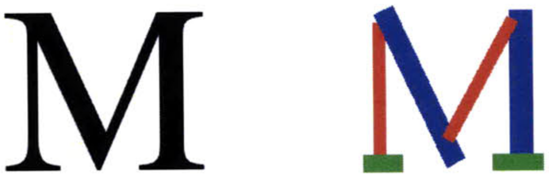


Figure 6.13: A character vectorized by many small rectangles.

Large cliques are often present when bonds are merged with a characters by noise in the underlying pixels. In addition, because cliques correspond to junctions, which are explicitly stored, finding cliques is a computationally simple problem: simply identify all the bonds that share a particular junction. Once the large cliques are found, the Vector Graph Builder assumes that all the single bonds in an image are the same size to within a constant factor between 1 and 2. It then reasons that if a single rectangle in the clique is significantly longer or shorter than the mode length of the rectangles in the clique, it is an out lier and should not be regarded as part of the clique. If the outlying rectangle is too short, it is simply erased. Short rectangles are the result of noise in the image or a bad vectorization. If it is too long, it is removed from the clique. Long rectangles are the result of bonds and characters being merged, and in this case the goal is to isolate the pixels that form the character and assert the possibility of this interpretation. In particular, if removing the rectangle from the clique results in the connected component of vectors that the clique belongs to being split into two connected components, the pixels corresponding to the remaining vectors in the clique are isolated and split off into a separate column. All the lower level senses are reapplied in the hope that now that the hypothesized merge between bond and character has been undone, the OCR subsystem will be able to recognize a character in the pixels. This method highlights one of the strengths of the nonlinear sense architecture; a high level sense made a decision that caused low level senses to re-evaluate the data in a new light that resulted in a global correction. This clique pruning technique has proven particularly useful in separating bonds from subscripts (Figure 6.14), where the vectors that approximate the subscript are small compared to the vector that approximates the bond.

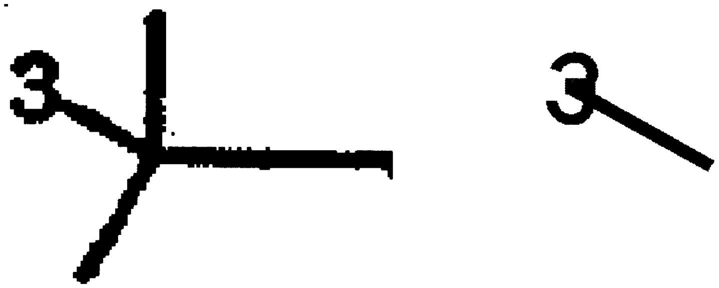


Figure 6.14: On the left are the pixels, on the right the vectorization. The blue bond is much larger than the nearby red and green vectors, so it is split away from them.

This step completes the initial vector graph and establishes a crude topology between vectors, and it marks a logical split in the reasoning about vectors. Until this point, there has been no notion of a bond, even though the heuristics discussed thus far have operated under the assumption that the underlying structure of the diagram is made of bonds and chemical formulas. All higher level senses, on the other hand, have some notion of a bond and explicitly use this information. In particular, there is the concept of a bond width and length estimate. The bond estimate is computed using the same method as the character width and height estimates described in chapter 3, except that rectangles that are significantly shorter than the average rectangle length are omitted from the calculation. This filtering prevents the constituent rectangles of hash bonds, that tend to be numerous and short, from skewing estimates that are intended to apply to a canonical standard single bond.

In addition, the notion of hash bonds, double, triple, and split bonds necessitate the notion of a vector sets; i.e., a set of rectangles that together define a bond. A vector set also has a bounding box that contains all the constituent rectangles; it can be viewed as the outline of the bond. The Vector Set Sense constructs these vector sets out of the rectangles, and addresses the aggregation problems described above in the list of tasks that the vectorization subsystem must accomplish. Bonds that consist of a single rectangle are also represented as vector sets for consistency; the sets contain one rectangle that is identical to the bounding box of the set.

I now describe how the Vector Set Sense solves the aggregation problems, and then I will explain how the vector sets are used to construct the skeleton of the final chemical graph. As it turns out, the aggregation problems for all types of bonds are variations on the hash bond aggregation problem, so we start with that. The problem has two parts; first, to identify those vectors that are potentially parts of hash bonds, and second, to decide if any given set of vectors forms a valid hash bond. Topological information is useful in both cases. Because a hash bond is made up of two or more isolated rectangles, vectors that have no neighbors in the graph are candidates to be part of a hash bond. Sometimes noise causes a hash bond vector to be merged with a nearby bond or character, so in addition to isolated vectors, the set of candidates also includes vectors with one neighbor in the graph, but that are significantly shorter than the estimated bond length.

In order to group the candidate rectangles into hash bond groups, they are first sorted by length so that the longest rectangles are considered first. Each rectangle is considered the base, or the longest rectangle, in a wedge-shaped hash bond. The hypothesized bond axis is taken to be the line perpendicular to the major axis of the rectangle. The midpoints of the major axes of all the other rectangles are projected onto bond axis. If the error vector is sufficiently small, they are included in a set of potential hash bond constituent rectangles. The constituent rectangles are then sorted by the signed distance of the midpoint projection along the bond axis. The sorted set is split into two sorted sets; those with distance less than zero, and those with distance greater than zero. The initial rectangle is added to each sorted set. For each of these sets, a hash bond test is applied that checks whether the set meets certain criteria concerning the shape that a hash bond must have, described below. If the set does not pass the test, the rectangle farthest from the initial rectangle is removed and the test is reapplied, until either the set passes the test, or there are fewer than two rectangles left in the set. Sets of rectangles that passed the hash bond test are then sorted by length so that the largest sets are first. The Vector Set Sense maintains a list of rectangles that have already been added to a vector set. It iterates over the list of sorted sets, checking that none of the rectangles in the set has already been committed to a vector set. If none have, it converts the sorted set into a vector set and adds the rectangles to the committed list. If some rectangles have already been committed, it removes these rectangles from the set, reapplies the hash bond test to make sure the updated set is still a potential hash bond, and if it passes the test reinserts the updated sorted set in the list of sorted sets to be processed. Because the updated set is smaller, it necessarily falls farther down in the list and will be revisited as the iteration continues. When the iteration is finished, all hash bonds that fit the criteria will have been identified. Figure 6.15 illustrates how the system identifies a set of rectangles that could be a hash bond. If it had been the case that the long single bond shown in Figure 6.15 (b) had been close enough to the dotted line, it would have been included in the set of rectangles. This is acceptable, because each set of rectangles is subjected to other tests to reject cases like this, discussed below.

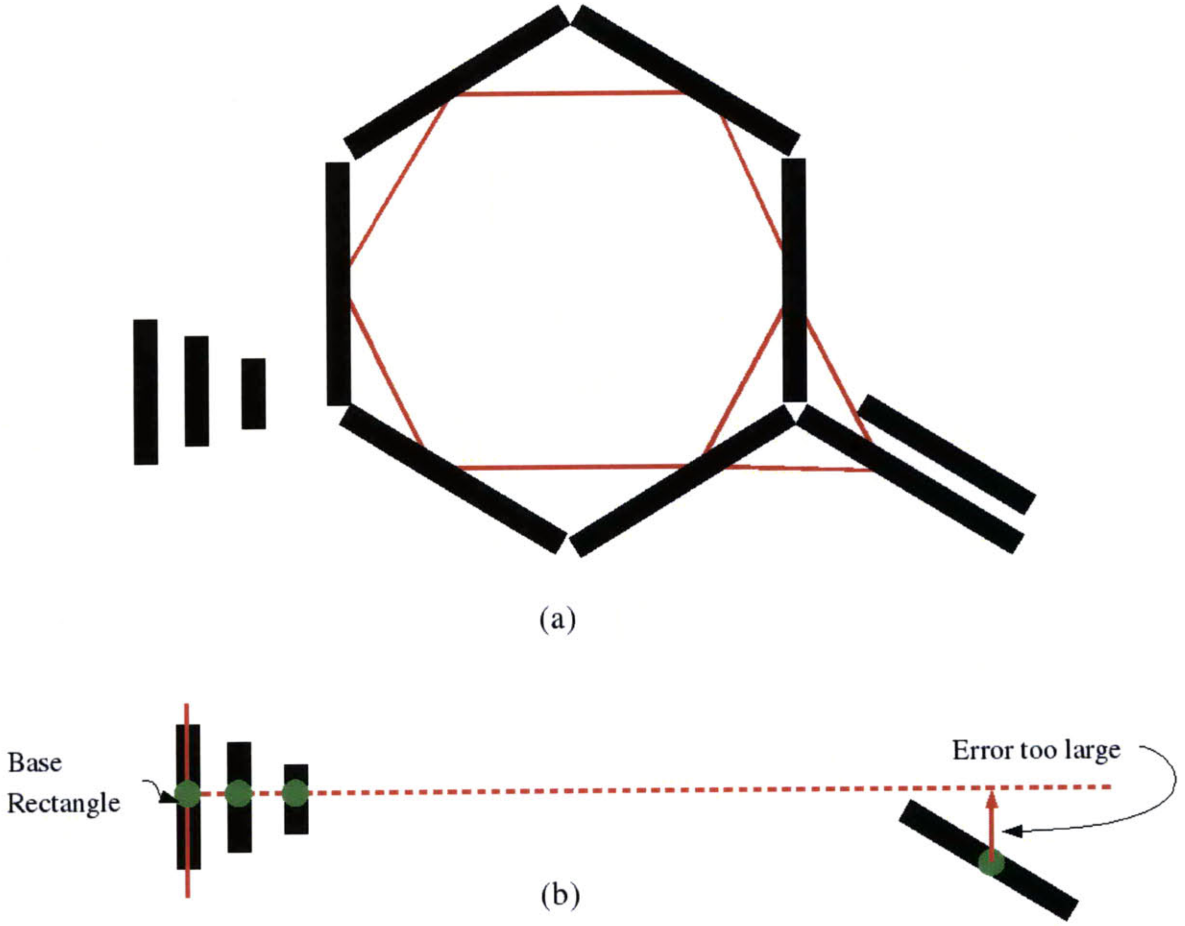


Figure 6.15: Hash bond rectangle aggregation. (a) shows the vectorization of an example molecule. The black rectangles make up the vectorization and the red lines indicate the topology. Rectangles connected by a red line are adjacent. The set of candidate rectangles, those that are isolated or have only one neighbor, is shown in (b). These rectangles are sorted by length, and each one is processed in turn. (b) shows the processing for the rectangle labeled Base Rectangle. The dashed line is the hypothesized bond axis, and the green circles are the midpoints of the major axes of the candidates. Rectangles whose midpoints project onto the bond axis with small error are included in a set that defines a potential hash bond. The rectangle on the far right is not included because the projection error is too large.

To be considered a hash bond, a set of rectangles must meet the following criteria: all the rectangles must be nearly parallel and shorter than the base rectangle to within a tolerance. The spacing between rectangles is constrained to be within a tolerance, as well as the variance of the spacing; rectangles that form a hash bond should be evenly spaced. In addition, the resulting bounding box for the set must be within a constant factor of the length and width estimates of a canonical single bond. The values of these constants are listed in the Appendix. Sets of rectangles that meet the criteria are collected into a vector set, which means that the pixels of all the rectangles in the set are collected together and a new column is defined over them. In this new column, the vectors are manually set to be the rectangles in the vector set, and OCR is explicitly disabled, signaling that this column absolutely does not correspond to a character.

This task exemplifies the flexibility of the column representation; columns make it possible to fluidly merge and split regions of pixels that logically belong together without expending a great deal of effort keeping track of the information each of the senses has contributed.

Notice that the rectangles that comprise double, triple, and quadruple bonds also meet both the hash bond criteria and the criteria to be included in the set of candidates. In this way, the hash bond aggregation procedure also aggregates these other types of bonds in most cases. The exception to this rule is shown in Figure 6.1 (c) (repeated below), where the rectangles of a single bond meet adjacent bonds at a junction. These double bond rectangles are excluded from the candidate set because they have too many neighbors, three neighbors in the case of Figure 6.1 (c). These types of bonds are handled at a later stage.

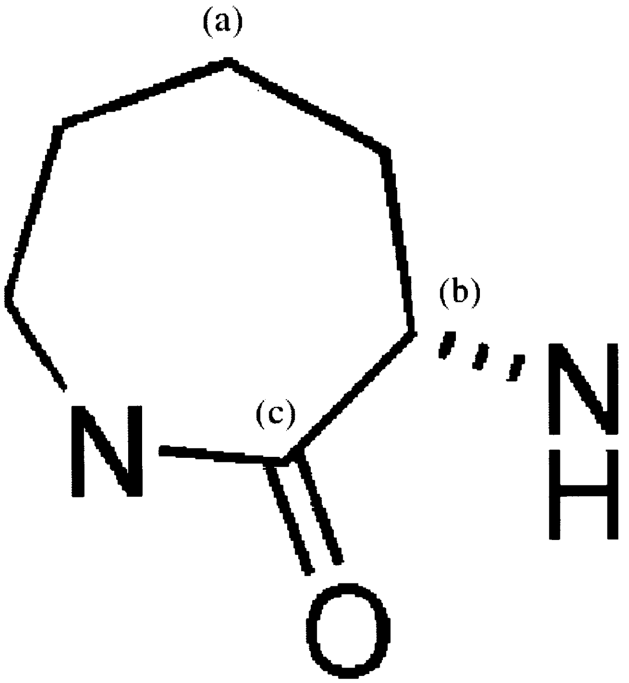


Figure 6.1: Topological challenges. In (a), the bonds overlap. In (b), the hash bond does not overlap the two adjacent standard single bonds. In (c), all four standard bonds share a single carbon atom, but they do not all overlap at the same point.

The one type of bond that is not covered by this case is the split bond. Split bonds are simpler than hash bonds because a split bond contains exactly two pieces. Candidate rectangles for split bonds are those with one junction or fewer, and do not already belong to another vector set. Pairs of candidate rectangles that meet the following criteria are placed in a vector set and marked as a split bond:

- Their long axes are nearly parallel.
- The line segment between the midpoints of the long axes of the two rectangles is nearly parallel to both axes. This is to prevent double bonds from being accidentally identified as split bonds.

- The bounding box of the two rectangles has width and length similar to the canonical single bond estimate.
- The line of pixels that falls on the long axis of the new rectangle is mostly dark.

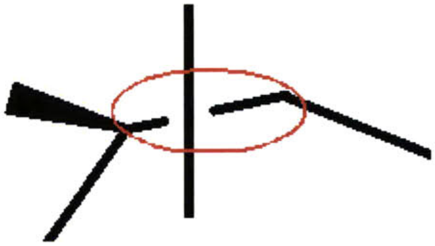


Figure 6.16: A typical split bond.

By this time, the aggregation phase is complete. A new bond width and length estimate is computed, and this estimate is used to further refine the vectors that are the only members of their vector sets. Rectangles that are very small compared to the new bond size estimate are deleted. All annuli are removed from connected components of rectangles in the vector graph that the OCR subsystem has not flagged as potential characters, because they are assumed to be bond blocks. This post processing completes the vectorization refinement phase, and the tasks delegated to the Vector Set Sense.

In order to produce the skeleton of the chemical graph, the Vector Set Graph Builder examines the vector graph and the columns that have vector sets. These columns contain two types of logical entities; connected groups of rectangles that correspond to bonds, and the disjoint rectangles that have either just been produced by the aggregation phase or are lone rectangles. The task of the Vector Set Graph Builder is to finally decide, as far as possible, which rectangles correspond to which bonds, and to convert the rectangles into an explicit bond representation. This is the final step of the vectorization refinement subsystem, and it is where the stereochemistry problem is at last solved. I will now explain how each type of entity is processed and added to the graph.

Connected groups of rectangles are processed first. Each rectangle in the group is considered to be either a wedge bond or a standard single bond. The rectangle, its vector set, and its pixels are converted into a new column. A stereochemistry detection routine is applied, which I describe momentarily. After all the rectangles have been converted into new columns, they are added to the chemical graph. Junctions from the vector graph are converted to explicit carbon atoms in the chemical graph, and rectangles that were adjacent in the vector graph are now explicitly connected to the carbon atoms that correspond to the junctions the rectangles shared. In this way, the chemical graph enforces the rule that bonds must always connect to atoms (or chemical formulas), and atoms (or formulas) must always connect to bonds. As bonds are added, if they have a single junction they are checked against the other bonds that share that junction. If the incoming bond is nearly parallel to an existing bond, the two columns are merged and the bonds are combined. This is how double bonds from Figure 6.1 (c) are identified.

We now turn to the stereochemistry detection algorithm. It must not only be able to detect wedge bonds, but also bold face bonds of the sort shown in Figure 6.3. To do this, it computes the sample mean and sample variance for the width of the bond. Wedge bonds will have a higher variance than bold and standard single bonds. Noise may also cause a high sample variance. If the variance is high, it checks whether the width is a certain percentage higher than the standard single bond width estimate computed by the Vector Set Sense. If so, the bond is classified as a wedge bond, otherwise as a standard single bond. If the variance is low, the bond may either be a bold or standard single bond, but it can not have a wedge shape. Again, the algorithm checks the sample mean width against the standard single bond estimate, but it uses a different threshold in order to classify the bond as a wedge bond. Note that semantically bold bonds and wedge shaped bonds are identical.

Rectangles that belong to vector sets that have been marked as split bonds are processed identically, except the bounding box of the set is used instead of the underlying rectangles, and a new column need not be created. They are then added to the chemical graph, and connected to the carbon atoms that correspond to the junctions that the underlying rectangles are associated with. This completes the processing for connected groups of bonds.

Sets of disjoint rectangles are more difficult to classify because context is often required. For example, the ambiguity surrounding a vector set containing a single rectangle has still not been resolved; it could be either an Iodine atom, or a single bond. Because the character aggregation subsystem has already been run by this point, the possibility that it is a lower case 'l' or the number '1' has been eliminated, because if it were either of these, the rectangle would have been aggregated into a chemical formula and removed from further consideration. When it is impossible to resolve the ambiguity, the Vector Set Graph Builder enables the OCR and constructs a distribution over what possible interpretations it believes the column could take, and passes the column up to a higher level sense. The Vector Set Graph Builder primarily makes use of the number of rectangles in the vector set when reasoning about the column. It can apply the following rules:

- If there are five or more rectangles, the set must be a hash bond.
- If there are four, three, or two rectangles, the OCR sense is enabled and special bond characters for quadruple, triple, and double bonds are added to the distribution. The hash bond character is also added.
- If there is a single rectangle, the Vector Set Graph Builder checks whether there is an existing bond in the chemical graph that is nearly parallel to this rectangle. If so, it checks that this rectangle could plausibly share the junctions associated with the parallel bond. If so, the rectangle is merged into the existing bond. This is how many double bonds are identified.

This completes the vectorization refinement subsystem. To summarize, it applies a series of heuristics to correct vectorization mistakes, group vectors that logically belong together, and convert sets of refined vectors to an explicit bond representation to the extent that is possible without context. By the time it is finished, the entire state of ChemWARD consists of the chemical graph, a set of aggregated chemical formulas, and any remaining unclassified columns that the Vector Set Graph Builder was unable to resolve. The Chemical Sense, described in the next chapter, constructs the context necessary to resolve the final ambiguities and completes the chemical graph.

Chapter 7

Using Chemical Knowledge

The chemistry subsystem is responsible for constructing the contextual information necessary to resolve any remaining ambiguous interpretations. It does this by carrying out the following three processes:

- Adding formulas to the graph and connecting lone bonds to the skeleton constructed by the vectorization refinement subsystem.
- Searching for unaccounted for regions of pixels throughout the image and suggesting chemically consistent interpretations.
- Searching for violations of valence constraints in the chemical graph.

These tasks are largely independent, and I describe each of them in turn.

As mentioned at the close of the previous chapter, the vector refinement subsystem forwards any unresolved columns to the chemistry subsystem. These consist primarily of bonds, but could also contain some unknown characters.

Remember that the chemical formulas constructed by the character aggregation subsystem have not yet been added to the chemical graph, which means that ChemWARD has not yet associated any bonds with any chemical formulas other than the implicit Carbon atoms that were created by the vector refinement subsystem. As a result, the examples of unresolved columns in Figure 7.1 might seem obvious to a human viewer, but are still ambiguous to ChemWARD. The first task that this subsystem accomplishes is to construct the context that makes the examples so obvious to a human viewer. In order to construct this context, ChemWARD adds the chemical formulas to the chemical graph, and thereby establishing the spatial arrangement between bonds and formulas.

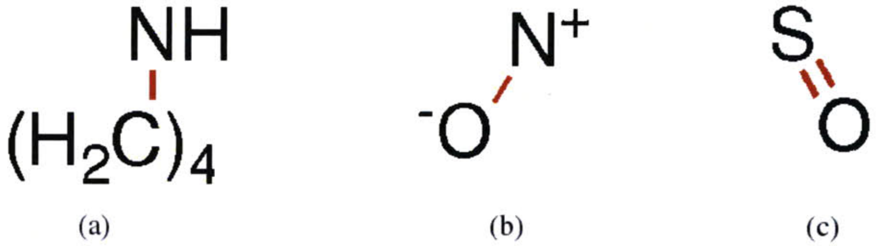


Figure 7.1: Ambiguous regions of pixels are highlighted in red. The chemical formulas shown in each case have not yet been added to the chemical graph. In (a) there is a rectangle that could be a single bond or it could be an 'I'. With context it is clear that it is a single bond, but that context has not yet been constructed; this is the job of the chemistry subsystem. In (b) there is a lone single bond. In (c) there are two rectangles that could either be a double bond or a hash bond. The location of the Sulfur and Oxygen atoms with respect to the rectangle make it clear that they form a double bond.

To add the formulas, ChemWARD computes a bounding box around each chemical formula and extends a margin around the bounding box. Bonds whose free ends fall in the margin of the bounding box are attached to the formula. ChemWARD seeks to avoid errors by enforcing that bonds don't connect to the same formula twice. Once this has been accomplished, adding the remaining entities is simpler because it is possible to reason about the location of formulas and bonds in relation to the entities being resolved.

ChemWARD resolves entities in an iterative fashion from the inside out. This means that in order to add an entity to the chemical graph, ChemWARD must identify either a formula or bond to attach the entity to. If it can not find anything close enough, ChemWARD places the entity at the end of the list of entities to process. This prevents entities from being added to the graph before their context has been added.

When adding entities to the graph, ChemWARD applies different resolution logic depending on the number of vectors in the vector set associated with each entity. If the entity in question is believed to be a non-bond character, the chemistry subsystem searches on the left and on the right for formulas to add the character to. If it finds formulas, the character aggregation subsystem is re-applied to the existing formula with the new entity appended (or prepended, or if the new entity forms a bridge between two formulas, on the whole new string of characters). If the new result obeys the syntax rules defined in the character aggregation subsystem, a new formula is produced and replaces the old one(s). Figure 7.2 shows an example; in this case the '2' forms a bridge between the red and blue formulas.

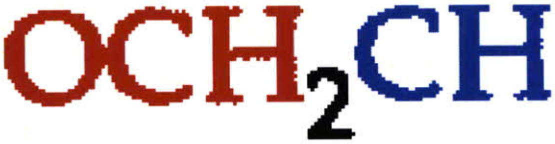


Figure 7.2: The '2' forms a bridge between the red formula and the blue formula.

If there are multiple vectors in the vector set associated with an entity, the chemistry subsystem searches for formulas in the directions parallel and perpendicular to the major axis of the bounding box of the set. The major axis of the bounding box will be either parallel or perpendicular to the direction of the major axes of the set of vectors. If formulas are found in the direction parallel to the major axes of the vectors, the entity must be a double, triple, or quadruple bond. If formulas are found in the direction perpendicular to the major axes, the entity must be a hash bond. This is how the double bond in Figure 7.1 (c) is resolved. If formulas are found in both directions, the chemistry subsystem prefers the formulas that are parallel to the bounding box of the set.

In the case of a single vector, there are two possibilities if the vector is a vertical rectangle. Either it is a single bond, or it is an Iodine. ChemWARD tests the Iodine hypothesis by converting the column into a chemical formula consisting of a single Iodine atom and searches for nearby bond endings as described above. If it finds any, it declares that the entity is an Iodine atom and inserts the new Iodine formula into the graph. Otherwise it assumes that the vector is a single bond, searches for nearby formulas, and if it finds any, converts the entity into a single bond.

Notice that the logic for resolving ambiguous entities is very simple because all the intermediate representations have already been computed and most of the low level work has already been done. Delaying the final decision until the last possible moment proves very effective when dealing with this type of inherent ambiguity found in molecular structure diagrams. Making these decisions at a lower level would have been error-prone. This is the real power of the architecture used in ChemWARD.

The second important task that the chemistry subsystem carries out is error detection and possibly correction. The chemistry subsystem traverses the final chemical graph and verifies that chemical constraints are met. The current implementation checks only two constraints. The first constraint is that each bond has either one or two junctions; any more or less indicates an error. A bond with more than two connections violates the definition of a bond: a connection between two atoms or compounds. A bond with zero junctions means that both of the atoms are implicit, and this can only happen C_2H_6. It is far more likely that zero junctions indicates that at least one of the formulas is missing. The second is that each formula connects to a plausible number of bonds. It is not possible to check this in all cases because it is not always possible to compute how many bonds a chemical formula can support due to the ambiguity inherent in chemical formulas (the same ambiguity that the search by structure problem attempts to alleviate). Nevertheless, where it is possible, the chemistry subsystem performs the verification. For example, a Nitrogen atom supports three bonds. Any more or less is a strong indication that there is an error.

When the chemistry subsystem detects a chemical inconsistency, it first tries to resolve it by searching in the nearby vicinity for unaccounted for regions of pixels. If it finds such a region, it re-applies the OCR and vectorization, and character aggregation subsystems. The result of this operation is that the region of pixels will have either been converted into a formula or it will have associated with it a distribution of bond and character hypotheses. If the chemical formula can support the required number of bonds to resolve the inconsistency, or any bond character in the set of hypotheses can resolve it, the chemistry subsystem resolves the new column to be that formula or bond, and adds it to the graph. This technique is useful for recovering regions of pixels that are very noisy and were rejected by other senses mistakenly. If the chemistry subsystem can not resolve the chemical inconsistency, it outputs a message to the user flagging the image for manual correction. Figure 7.3 shows an image where the chemistry subsystem might correct an error by suggesting that the highlighted region of pixels is an atom that supports 3 bonds.

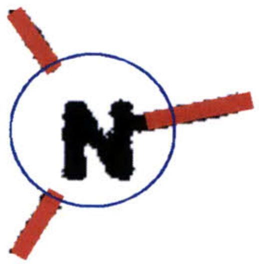


Figure 7.3: The black region of pixels is unaccounted for. The chemistry subsystem notices the region, and the three nearby bonds, and will prefer an interpretation that is a character that supports three bonds.

While the chemistry subsystem (Figure 7.4) performs important resolution logic, it does not make use of many chemical heuristics. I believe that ChemWARD could be improved by expanding the chemistry subsystem. Other limitations of ChemWARD and directions of improvement are discussed in chapter 8.

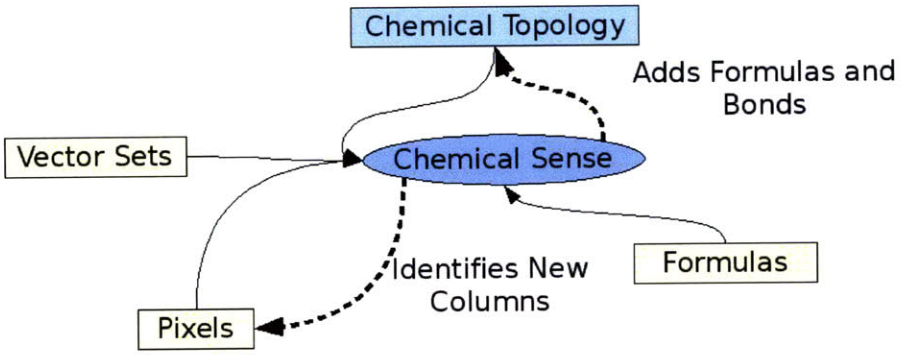


Figure 7.4: The chemistry subsystem. Solid lines indicate that the chemical sense examines the data in the intermediate representation. Dashed lines represent modifications, which are labeled.

Chapter 8

Limitations and Future Work

ChemWARD is a work in progress. In this chapter, I describe the limitations the program is currently subject to, and directions of future work on the molecule recognition problem. Figure 8.1 repeats ChemWARD's architecture diagram (originally shown in Figure 2.6) with the subsystems discussed in previous chapters highlighted. I present in this chapter the limitations in each subsystem from the bottom up, and then the limitations of ChemWARD as a whole.

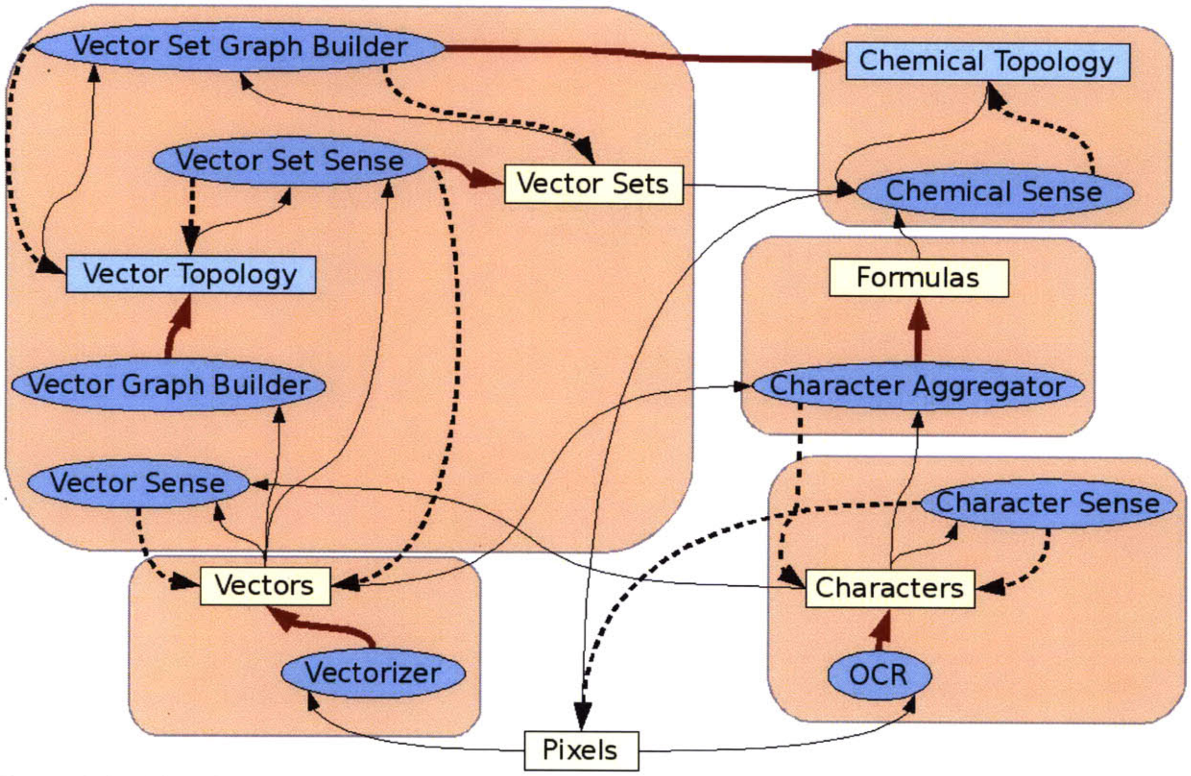


Figure 8.1: ChemWARD's architecture with each subsystem highlighted.

The lowest level subsystems concern OCR and vectorization, and so I address those first. Both of them have significant room for improvement, but for very different reasons.

As discussed in chapter 3, the choice of OCR engine in ChemWARD was influenced by reasons other than performance. SimpleOCR was the best OCR engine I could find that was within budget and provided access to the source code for integration into ChemWARD. It uses a neural network for character recognition. Accuracy suffers in noisy images, and the OCR engine sometimes very confidently makes mistakes when generating hypotheses for characters. In the current system, if the OCR engine does not produce the right hypothesis for a character, ChemWARD will almost certainly make a mistake in the final interpretation.

Consequently, in order to improve ChemWARD's performance, a better OCR engine should be found or built. Also mentioned in chapter 3, the current state of the art in OCR technology exists in industry and usually is not available for modification and extension, so constructing a custom OCR engine will be a time consuming and challenging task. Nevertheless, I believe it is the best solution for many reasons. It will be possible to build into the OCR engine itself recognition procedures for bonds instead of relying on external senses. This will improve performance because it will reduce the number of false hypotheses that another OCR engine might generate when presented with a bond. In addition, building better routines to recognize subscripts and superscripts would be possible. Another advantage is that in a custom OCR engine it would be possible to provide explicit suggestions concerning which character a region of pixels could be. For example, the Chemical Sense might identify an inconsistency in the diagram and formulate an expectation about what character would resolve it. It could then explicitly query how confidently the OCR engine believes the region is that particular character given that the Chemical Sense believes it is likely to be that character. To my knowledge, in most OCR engines this is impossible. Finally, where many OCR engines rely on dictionary files and clustering to improve performance, perhaps trading off accuracy for individual characters, a custom OCR engine could be tuned to focus entirely on individual character recognition.

The vectorization subsystem has a different limitation. For both rectangles and annuli, the scoring functions are high dimensional, and the space that they optimize over is possibly discontinuous. While the vectorization produced by the current version is not poor, chapter 6 described a very elaborate system of refinement that is necessary to extract from the vectorization the bond information that it is designed to produce. I believe there are two principle directions in which to improve the quality of vectorization. The first is to simplify the scoring functions by reducing dimensionality. This would make the optimization space easier to reason about, and make the output more predictable. In addition, setting the weights in the scoring functions proved difficult because I did not have time to set up a rigorous training system. Producing a training system will almost certainly make the scoring functions more effective.

The second direction of improvement is specific to annuli. As mentioned in chapter 5, identifying annuli is useful in segmenting merged characters and bonds because an annulus is indicative of the presence of something other than a bond. At the moment, ChemWARD is able to recognize only circular annuli. In many fonts, the arcs in characters are elliptical rather than circular. The result is that ChemWARD often approximates a single elliptical annulus with two circular annuli, or one circular annulus and one rectangle. In order to avoid this noisy result, ChemWARD's vectorization algorithm should be extended to recognize elliptical annuli. This extension is straightforward for the discrete gradient ascent part of the vectorization algorithm, because the only requirement is a scoring function and a set of allowable transformations that can be applied to the current best fit. It is more difficult to produce initial guesses for annuli. Because the general equation for an ellipse has terms that contain both x and y, using the current method of defining another dimension is not applicable. But, the equation for an axis aligned ellipse has no cross terms, so it is in principle possible to modify the current arc generating routine to generate axis aligned elliptical arcs. Due to time constraints, I was not able to make this extension, but the next iteration of ChemWARD would not be complete without it.

Above the vectorization subsystem is the vectorization refinement subsystem. Because it consists entirely of heuristics, it is difficult to make precise claims about how to improve it. Nevertheless, every heuristic requires one or more parameters, all listed in the Appendix. As was the case with the vectorization algorithm itself, these parameters are my coarse estimates and not the product of a formal training system. Constructing a training system may improve results on this front, but I believe that part of the value of these heuristics is that they do not need precise parameter settings in order to be effective. If they did, the system would be much more fragile, and would fail when faced with images in a slightly different style.

Continuing on, the character aggregation subsystem is next. Because the character aggregation system is relatively simple, it is not a major limitation in ChemWARD. In terms of performance, there are two possible improvements. At the moment, the aggregation and resolution steps described in chapter 4 are interleaved; that is, the characters in a group are resolved as soon as the group is identified. I believe a better approach would be to aggregate everything first, then resolve the characters. The reason this approach is better is that in the resolution phase, each group is tokenized into superscripts, characters, and subscripts. As mentioned in chapter 4, this is accomplished using simple distance thresholds. I believe it would be better to try to learn where the distance thresholds are by examining all the groups at once, and only after some reasonable estimate of the thresholds has been observed and calculated tokenize the groups of characters. This strategy would make the aggregation phase more robust to different styles of images.

In addition to improving tokenization, there are syntactic conventions that the character aggregation subsystem simply does not support at this time. These are principally nested parentheses and dashes as part of chemical formulas (i.e., "n-Bu"). They were not implemented due to time constraints.

Finally, we come to the chemical system. As mentioned in chapter 7, it turned out that drafting knowledge is far more important than chemistry knowledge. Consequently, the chemistry subsystem is used mainly to identify inconsistencies in the output produced by ChemWARD. I believe the most interesting direction in which to expand this subsystem is to try to take advantage of weak indicators like symmetry, in order to direct the other senses to look more closely at certain regions of the image that they would make mistakes on otherwise. For example, Figure 8.2 shows a molecule where a bond is so small that a human viewer might also miss it. ChemWARD does in fact reject the highlighted bond as noise. Nevertheless, a human viewer will immediately recognize the symmetry in the bond structure and conclude that the small region of pixels that does not otherwise look like any bond in the image is actually a bond. Of course not every molecule will have such symmetry and some may be interesting only because they are asymmetric. This is why weak indicators should only be used to direct other senses, not to make hard conclusions.

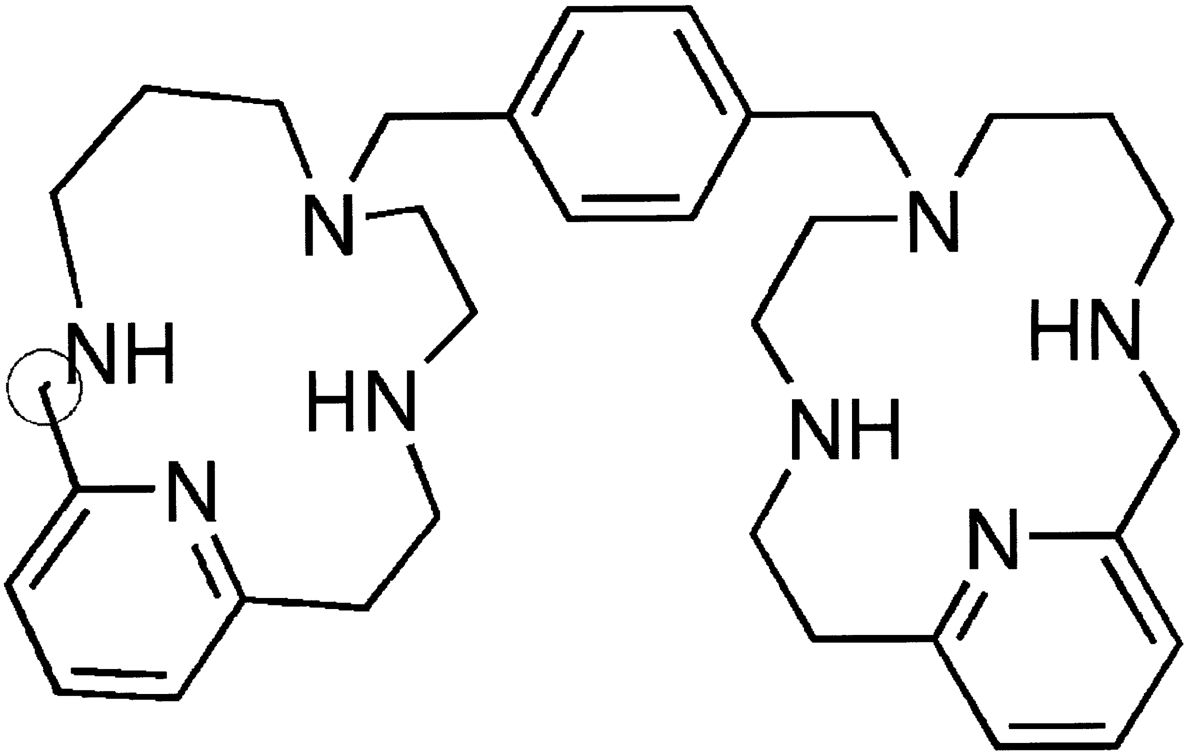


Figure 8.2: The bond highlighted in red is much smaller than any other bond in the image, and might legitimately be considered noise. It is the symmetry of the molecular structure that indicates that it is definitely a single bond.

Beyond the limitations within each subsystem, there are more fundamental limitations that ChemWARD is subject to as a whole, that can not be overcome by improving only one subsystem. These are support for drafting conventions, support for different output formats, extracting images from documents (as opposed to extracting molecular structure from images), and speed. I address each of these in turn.

ChemWARD does not support a number of common conventions used in molecular structure diagrams. These include wavy bonds, bonds that are wedge bonds but not single bonds, circles around charges, aromatic rings, and Markush structures (Figure 8.3). I did not have time to implement support for these conventions.

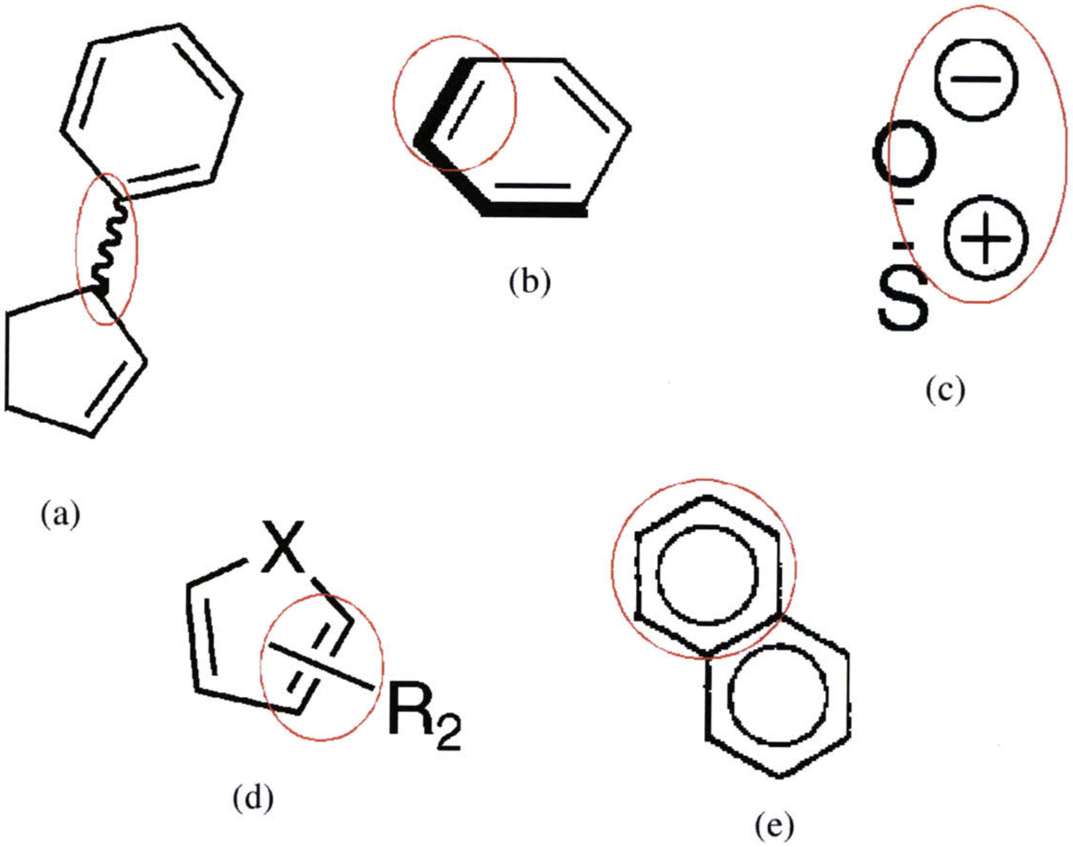


Figure 8.3: Conventions not supported in the current version of ChemWARD. (a) is a wavy bond, indicating ambiguous stereochemistry. (b) is a double bond that is directed out of the page. In (c) there are circled charges associated with the oxygen and sulfur ions. (d) shows a Markush structure, and (e) shows two aromatic rings.

In the current incarnation, ChemWARD generates a graph of chemical formulas and the bonds that connect them, capturing both geometric and topological information. However, in order to create the search index that is the ultimate goal, ChemWARD should be able to convert the chemical graph into standard output formats such as SMILES, Mol, InChi, etc. Again, time was the limiting factor. A future version of ChemWARD will include some or all of these formats.

While ChemWARD is designed specifically to solve the molecule recognition problem, a solution to the search by structure problem described in chapter 1 must have some related functionality. In particular, automatic extraction of structure diagrams from the documents they are embedded in is to my knowledge a problem without a good solution. This problem is difficult because supporting text often appears within the bounding box of the molecular structure diagrams, e.g. Figure 8.4, including page numbers, reactions, and annotations. One solution is to attempt to recognize and eliminate this text. Another system [9] interprets the text to find the identity of shorthand that might be used in the molecular structure diagrams themselves, i.e., R-groups. I believe this is a profitable direction for expansion that will contribute to any solution for the search by structure problem.

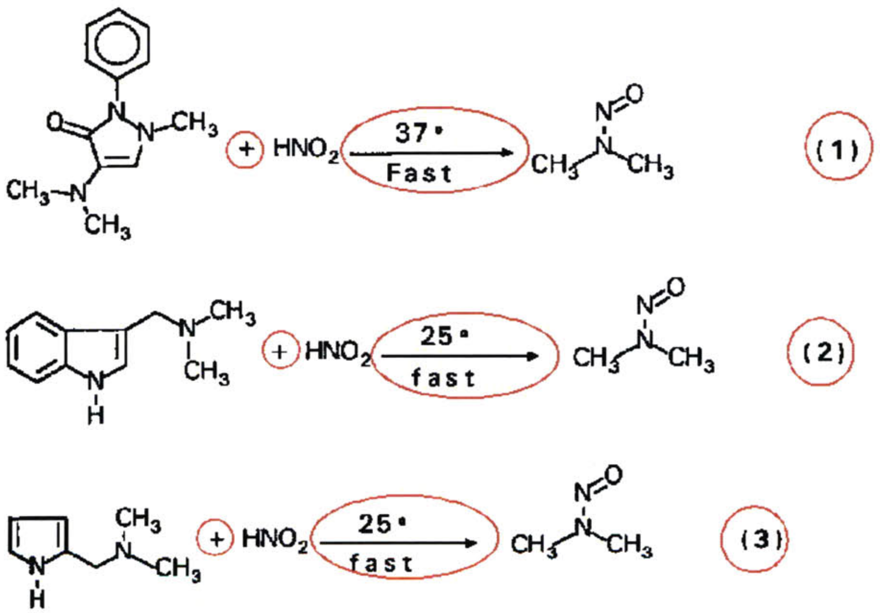


Figure 8.4: Molecular structure diagram with supporting text (highlighted in red).

A final topic to address is speed. Speed is not critical for an application that processes structure diagrams offline, so I did not attempt to optimize any of the code for speed. In this first version, I am primarily concerned with accuracy and designed. Nevertheless, it is possible to improve speed in nearly every part of the code, but perhaps nowhere more so than in the vectorization algorithm. As described, every rectangle or ellipse must be rasterized in order to be scored. When searching for better fits, the shapes that are generated are only incrementally different from the current best fit, so it is possible to re-use most of the previous rasterization. At the moment, ChemWARD makes no effort to do this, and rasterization as part of vectorization takes up nearly half of the time in computation.

In this chapter I presented ChemWARD's limitations and steps to work towards overcoming them. In the next chapter, I compare the current incarnation of ChemWARD to other systems that attempt to solve the molecule recognition problem.

Chapter 9

Results

This chapter compares ChemWARD to other systems that solve the molecule recognition problem. It highlights some of the differences between ChemWARD and other systems, then it describes the set of images used when measuring performance. It discusses the standards of performance considered in ChemWARD, and finally, it analyzes ChemWARD's performance on the image set.

It is difficult to provide an exhaustive list of the differences between ChemWARD and other molecule recognition systems because the papers that describe them do not include all the implementation details, so I don't attempt to do this here. The descriptions I provide are intended to be general, and the reader should refer to the references provided for each of the systems discussed for more complete information. I compare ChemWARD to three other molecule recognition systems; CLiDE [7], ChemOCR [8], and OSRA [9].

CLiDE is a well-established molecule recognition system that has been intermittently under development for a number of years. The most recent iteration was published in March, 2009 [7]. CLiDE has the advantage that it addresses a number of issues peripheral to the molecule recognition problem that are necessary in order to solve the search by structure problem. In particular, CLiDE performs document analysis to extract the molecular structure diagram from the document it appears in. Also, CLiDE can generate Molfiles, which is not yet supported in ChemWARD. It can recognize wavy bonds, dashed bonds, and aromatic rings, none of which is currently supported in ChemWARD. In terms of implementation, CLiDE uses a contour following vectorization algorithm, and a modified OCR engine that is not described in detail in the paper. It then employs a number of bond recognizers to identify the different types of bonds.

ChemOCR, on the other hand, employs an expert system to classify regions of pixels as chemical formulas or different types of bonds. In order to solve the aggregation problem, it employs a neighborhood graph that defines a distance metric between regions of pixels, and it applies the rules in the expert system to groups of nearby regions of pixels. Conflicts between rules in the expert system are resolved after all the rules have been applied. I believe that a flaw in this system is that all the rules are applied at once; there is no real notion of a hierarchy of levels of interpretation. ChemOCR also employs its own vectorization algorithm that is not described precisely in the publication.

The third system, OSRA, is unique in that it is composed entirely of open source tools. For vectorization it uses the Potrace algorithm [21], and for OCR it uses both GOCR [13] and OCRAD [26], and then employs a number of recognition procedures to identify the different types of bonds. OSRA is capable of recognizing some conventions that ChemWARD does not support, such as aromatic rings and dashed bonds.

In measuring performance, ChemWARD uses a set of images published by CLiDE, and used as the test images for the current version of CLiDE. The images are not very noisy, but are large and have complex geometry. Because ChemWARD does not support extracting the image from the document, I did this manually. I removed from the image set any page numbers or text that was not part of the molecular structure, and I placed each molecular structure in a separate file. I also converted any images that were not bitmaps to bitmaps. In addition, any images that contain conventions not supported by ChemWARD were excluded from the sets used to benchmark ChemWARD's performance. The images were not modified in any other way. The image set contains 483 molecular structures.

Because each of the systems mentioned uses a slightly different set of images to measure performance, the comparison presented here is not definitive. The comparison with CLiDE is the most informative because I used a subset of the CLiDE images to test ChemWARD.

I used two accuracy measures. The stricter one counts a diagram as interpreted correctly only if the interpretation is perfect. A second metric analyzes each recognition in each diagram and considers what percent of each recognition type (characters, bonds, etc.) was performed correctly.

Because the molecule recognition problem is solved off line, I have not considered performance metrics such as run time or memory usage. While computational performance is an important practical concern, the current iteration of ChemWARD is concerned only with recognition rate.

The results are as follows:

Table 1: Perfectly Recognized Molecules

System:	Total Molecules:	Perfect Molecules:	Percent:
ChemWARD	483	413	85.5
CLiDE Pro	519	466	89.79
OSRA	215	107	49.77
OSRA	42	26	61.90
ChemOCR	100	72	72.00

Table 2: Error Analysis

Feature:	Instances:	Mistakes:	Percent:
Character	5814	35	99.4
Superscript	54	14	74.1
Subscript	312	1	99.7
Charge	47	41	87.2
Bond	15193	115	99.2
Wedge Bond	291	9	96.9

Noise Region	-	33	-
Other	-	413	-

When reading the error analysis, there are two considerations to keep in mind. First, the error categories often overlap. For example, an interpretation could be missing a subscript, and the region of pixels that corresponds to the missing subscript could be rejected as noise, resulting in a mistake in both the Subscript and the Noise category. Second, when counting errors, I did not include false positives in each of the categories with a percentage. The percentage column reflects the fraction that was correctly detected. False positives are counted in the Other category.

ChemWARD produces more accurate results than all other systems except CLiDE. This result shows that the approach is sound but the system needs improvement. As mentioned several times, ChemWARD has many parameters that were determined empirically. I believe that setting up a rigorous training harness to tune these parameters will significantly improve performance by reducing the number of false positives.

Chapter 10

Contributions

The most important contribution ChemWARD has made to the molecule recognition problem is the set of general principles mentioned in chapter 2:

- Use high level information to correct low level mistakes.
- The further a mistake propagates, the more difficult it is to correct.
- It is better to remain uncertain about an interpretation and maintain ambiguity than to commit early to a mistake.
- It is easier to rule out possible interpretations of a symbol than to commit to one.

While these ideas are not new in and of themselves, by systematically incorporating all of them, ChemWARD produces performance that is competitive with the state of the art. By continuing to make use of these guiding principles, ChemWARD can be improved and expanded, ultimately to help solve the search by structure problem.

A second important contribution is that ChemWARD clearly defines the multiple levels of interpretation involved in extracting molecular structure, and employs an architecture that allows free flow of information between them, while keeping complexity at a manageable level. While this idea too is not new, it is my belief that such an architecture is absolutely crucial to solving the molecule recognition problem; ChemWARD serves as a proof that that technique is applicable in this domain.

In particular, this technique allowed the implementation of image segmentation heuristics for separating bonds and characters, which has proven to be a challenging task that neither OCR engines nor vectorization algorithms can accomplish well independently.

ChemWARD has also led to the novel vectorization algorithm described in chapter 5.

Just as important as the optimization technique is the metric for what is a good vectorization; namely the bias towards extracting the types of shapes found in molecular structure diagrams. By vectorizing annuli as well as rectangles, the vectorization algorithm produces fewer vectors and exposes a more accurate picture of the underlying shapes in the image.

Finally, ChemWARD is a working end to end system that can be modified and improved by future researchers on the molecule recognition problem.

Appendix

Listed below are ChemWARD's system parameters, along with a concise reminder about what they are for. They are organized by subsystem, starting at the lowest level.

Vectorization: The weights used in the vectorization scoring function are as follows:

For rectangles:

50.0	Boundary Pixels
40.0	Interior Pixels
-50.0	Missing Boundary Pixels
-120.0	Missing Interior Pixels
5.0	Duplicate Boundary Pixels
20.0	Duplicate Interior Pixels
-100.0	Duplicate Missing Boundary Pixels
-100.0	Duplicate Missing Interior Pixels
-10.0	Set Pixel Count
10.0	Missing Pixel Count
-25.0	Penalty on the normalized dot product between the major and minor axes
2.0	Bias On Major Axis
1.0	Bias On MinorAxis

For Annuli:

50.0	Boundary Pixels
40.0	Inerior Pixels
-50.0	Missing Boundary Pixels
-200.0	Missing Interior Pixels
25.0	Duplicate Boundary Pixels

20.0	Duplicate Interior Pixel
-200.0	Duplicate Missing Boundary Pixels
-200.0	Duplicate Missing Interior Pixel
-10.0	Set Pixel Count
10.0	Missing Pixel Count
10.0	Bias On [Outer Radius – Inner Radius]

Character Sense: Let the character height estimate be H_C and the character width estimate be W_C. Let the height of a region of pixels be h, and the width be w. All regions of pixels that correspond to characters must have $A * H_C < h < B * H_C$, and $C * W_C < w < D * W_C$. The Character Sense attempts to segment regions that are too wide.

A	.4
B	1.75
C	.9
D	1.6

Character Aggregation: Let Y be the minimum y coordinate of the first character F in the character aggregation process. Characters can be aggregated with F only if their minimum x coordinates fall within $Y - A * H_C$ and $Y + H_C$. Each successive character's minimum x coordinate must be within $B * W_C$ of the previous character's maximum x coordinate. Lines of characters are combined if the minimum x coordinates of the first characters' of the lines are within $C * W_C$ of each other, and the vertical empty space between them is less than $D * W_C$.

A	.6
B	.75
C	.125

D	.5

Vectorization Refinement:

Rectangles: If two rectangles share more than A% of their area, or if they share more than B% of their border area, they are merged. If the angle between the major axes of two rectangles is less than C degrees, they are merged. If a single rectangle covers more than D% of the pixel region it is a part of, the pixel region is replaced by the bounding rectangle of the region.

A	50
B	25
C	7.5
D	75

Annuli: If the distance between the center of two annuli is less than A pixels, and the difference between the maximum inner radius and the minimum outer radius is less than B pixels, the annuli are merged.

A	1.73
B	1.41

Topological Criterion 2: The projection error of the endpoint of one rectangle onto the medial axis of the other must be less than 50% of the length of the medial axis being projected onto in order to construct a junction between the two rectangles.

Let the bond length estimate be L_B and the bond width estimate be W_B.

Junction Aggregation: When merging nearby junctions, the new junction must be within 25% of LB from each of the old junctions.

Clique Pruning: For all cliques of 4 vectors or more, if a single vector is longer than A * the mode of all the vector lengths in the clique, or shorter than B * that same mode, it is removed from the clique and the pixels are split.

A	2.4
B	.5

Hash Bond Aggregation: In order for a rectangle to be considered a candidate for a hash bond, it must be shorter than A * LB. When forming initial sets of rectangles, the projection error of the midpoint onto the axis perpendicular to the major axis of the base rectangle must be less than B pixels.

Each set of rectangles is a hash bond if it meets the following:

- The maximum angle between the major axes of all the rectangles is less than C degrees.
- The maximum difference between the lengths of the longest and shortest rectangles is less than D * LB.
- The maximum difference between the widths of the widest and narrowest rectangles is less than E * WB.
- The maximum spacing between rectangles must be less than F * LB.
- The sample variance of the spacing must be less than G.

A	.8
B	7
C	12.5

D	.25
E	4.5
F	.5
G	16.0

Split Bonds: Pairs of rectangles are split bonds if they each have no more than one neighbor in the vector graph and they meet the following criteria:

- The angle between their major axes is less than A degrees.
- The ratio between the sum of the lengths of the rectangles and the length of the bounding rectangle is B
- The length of the bounding rectangle is between $C * L_B$ and $D * L_B$.
- The width of the bounding rectangle is between $E * L_B$ and $F * L_B$.

A	15.0
B	.6
C	.5
D	1.75
E	.5
F	3.0

Stereochemistry: Let w be the average width of a bond, and let s be the sample variance of the width. If $s > A$, the bond is a wedge bond if $w > B * W_B$. Otherwise, the bond is a wedge bond if $W > C * L_B$.

A	.1
B	1.125
C	2.0

Constructing the chemical graph: If two bonds share a junction and the angle between their major axes is less than 5 degrees, they are merged.

References

[1] T. Y. Ouyang, "Recognition of Hand Drawn Chemical Diagrams," Cambridge: Massachusetts Institute of Technology, 2007.

[2] D. Tenneson, "Interpretation of Molecule Conformations from Drawn Diagrams," diss, Providence: Brown University, 2008.

[3] S. Mori, C. Y. Suen, and K. Yamamoto, "Historical Review of OCR Research and Development," IEEE vol. 80, no. 7, Jul., pp 1029-1058, 1982.

[4] M. H. Coen, "Multimodal Dynamics: Self-Supervised Learning in Perceptual and Motor Systems," diss, Cambridge: Massachusetts Institute of Technology, 2006.

[5] L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy, "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty", AMC Computing Surveys, vol 12, no. 2, Jun., pp. 213-253, 1980.

[6] P. Ibison, M. Jacquot, F. Kam, A. G. Neville, R. W. Simpson, C. Tonnelier, T. Venczel, and A. P. Johnson, "Chemical Literature Data Extraction: The CLiDE Project," Journal of Chemical Information and Computer Science, May, pp. 338-344, 1993.

[7] A. T. Valko, and A. P. Johnson, "CLiDE Pro: The Latest Generation of CLiDE, a Tool for Optical Chemical Structure Recognition," Journal of Chemical Information and Modeling, Apr., pp. 780-787, 2009.

[8] P. Kral, "Chemical Structure Recognition via an expert system guided graph exploration,", Munich: Ludwig-Maximilians-Universitat Technische Universitat Munchen, 2007.

[9] I. V. Filippov, and M. C. Nicklaus, "Optical Structure Recognition Software To Recover Chemical Information: OSRA, An Open Source Solution," Journal of Chemical Information and Modeling, Mar., pp 740-743, 2009.

[10] ABBYY – OCR, ICR, OMR, Data Capture and Linguistic Software. ABBYY. 07 Sept. 2009 <http://www.abbyy.com>.

[11] OCR Xpress Professional. Component Source. 07 Sept. 2009 <http://www.componentsource.com/products/ocr-xpress-professional/index.html>.

[12] Readiris 12 – OCR Software. I.R.I.S. 07 Sept. 2009 <http://www.irislink.com/c2-1584-225/Readiris-12---OCR-Software-------Convert-your-Paper-Documents-into-Editable-Text-.aspx?adwp=GGS-RI&gclid=CILN1tOt4ZwCFRZeagod7VmPmw>.

[13] GOCR. 07 Sept. 2009 <http://jocr.sourceforge.net/>.

[14] tesseract-ocr. Google. 07 Sept. 2009. <http://code.google.com/p/tesseract-ocr/>.

[15] SimpleOCR. SimpleSoftware. 07 Sept. 2009. <http://www.simpleocr.com/>.

[16] Scan to PDF Software. Softi Software. 07 Sept. 2009. <http://softi.co.uk/freeocr.htm>

[17] PdfCompressor. cvision. 07 Sept. 2009.

<http://www.cvisiontech.com/products/general/pdfcompressor.html>

[18] H. Tamura, "A Comparison of Line Thinning Algorithms From Digital Geometry Veiwpoint," In Proc. Fourth Int'l Joint Conf. Pattern Recognition, 1978, pp. 715-719.

[19] D. Dori, and W. Liu, "Sparse Pixel Vectorization: An Algorithm and Its Performance Evaluation," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 21, no. 3, Mar., pp 202-215, 1999.

[20] J. Song, F. Su, J. Chen, C. Tai, and S. Cai, "Line Net Global Vectorization: an Algorithm and Its Performance Evaluation," Nanjing University, 2000.

[21] P. Selinger, "Potrace: a polygon-based tracing algorithm," [Online document], 2003 Sept. 20, [cited 2009 Sept. 7] Available HTTP: http://potrace.sf.net/potrace.pdf.

[22] D. Elliman, "A Really Useful Vectorization Algorithm," Lecture Notes In Computer Science, vol. 1941, pp. 19-27, 1999.

[23] O. Hall-Holt, M. J. Katz, P. Kumar, J. S. B. Mitchell, and A. Sityon, "Finding Large Sticks and Potatoes in Polygons," In Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms, 2006, pp. 474-483.

[24] R. Fruhwirth, A. Strandlie, W. Waltenberger, and J. Wroldsen, "A review of fast circle and helix fitting," Nuclear Instruments & Methods in Physics Research, vol. 502, pp. 705-707, 2003.

[25] G. Toussaint, "Solving Geometric Problems with the Rotating Calipers," In Proc. IEEE MELECON, 1983.

[26] OCRAD. 07 Sept. 2009 <http://www.gnu.org/software/ocrad/ocrad.html>.