

Recognizing Free-form Hand-sketched Constraint Network Diagrams by Combining Geometry and Context

Tracy Hammond¹ and Barry O’Sullivan²

¹ Sketch Recognition Lab, Department of Computer Science, Texas A&M University, College Station, Texas, 77843, USA, hammond@cs.tamu.edu

²Cork Constraint Computation Centre, Department of Computer Science, University College Cork, Ireland, b.osullivan@cs.ucc.ie

Abstract

Constraint satisfaction problems (CSPs) are ubiquitous in many real-world contexts. However, modeling a problem as a CSP can be very challenging, usually requiring considerable expertise. In many application domains there can often be a domain-specific way of drawing a graphical representation of a problem. Our objective is to develop sketch recognition technology that can recognize hand-drawn representations of problems, and automatically generate constraint satisfaction models of them. This paper describes a sketch recognition system that recognizes and solves a simplified set of hand-drawn constraint problems. Shapes are recognized using a combination of geometric and contextual rules, allowing shapes to be drawn freely, without requiring a specific drawing style.

Categories and Subject Descriptors (according to ACM CCS): I.7.5 [Document Capture]: Graphics recognition and interpretation H.5.2 [User Interfaces]: Input devices and strategies, Interaction styles I.5.5 [Implementation]: Interactive Systems.

1. Introduction

Computers play an increasingly important role in helping individuals and industries make decisions. For example, they can help individuals make decisions about which products to purchase or industries make decisions about how best to manufacture these products. Difficult problems can offer too many choices, many of which are incompatible, few of which are optimal. Constraint programming makes it easier for computers to help us make these choices. For example, the problem of scheduling employees to shifts is a constraint satisfaction problem. Constraints may range from governmental regulations on working hours to the specialized skills of individual workers. Constraints can also take into account costs, preferences, priorities and uncertainties. For example, a worker may prefer a certain shift, an employer may seek to provide sufficient coverage at minimal cost. Constraint optimization methods can obtain satisfactory or ideally optimal solutions. Constraints arise in design and configuration, planning and scheduling, diagnosis and testing, and in many other contexts. Constraint programming can solve problems in telecommunications, internet commerce, bioinformatics, transportation, network management, manufacturing, supply chain management, finance, and many other fields [Wal96].

Constraint programming already has wide commercial application, but much remains to be done to fully explore and

exploit the technology. Constraint programming is studied in artificial intelligence, as constraint-based reasoning or the study of constraint satisfaction problems (CSPs), as well as in other disciplines, especially through the study of specialized constraint programming. Constraint programming and optimization provide powerful support for decision-making; it is able to search quickly through an enormous space of choices, and infer the implications of those choices.

However, the impact of constraint programming has itself been constrained by a lack of “user-friendliness”. To some degree user-friendliness could be regarded as the 21st century challenge for all of computer software, but constraint programming presents particular challenges and opportunities. Jean-Francois Puget, Vice President of Optimization R&D at ILOG, delivered an invited talk at the 2004 Constraint Programming conference: “Constraint Programming’s Next Challenge: Simplicity of Use”. The objective of the work presented in this paper is to enhance the usability and applicability of constraint programming and optimization by providing intelligent sketch-based interfaces that allow individuals to hand-sketch these problems on a TabletPC as they would on paper, while the computer observes and understands these diagrams, solving and simplifying the diagrams with powerful constraint satisfaction tools.

Constraint programming can be divided very crudely into

modeling and solving. Modeling defines the problem, in terms of variables that can take on different values, subjects to restrictions (constraints) on which combinations of variables are allowed. Solving (often using inference and search) finds values for all the variables that simultaneously satisfy all the constraints. In the work presented here we emphasize the modeling aspect of constraint programming. It is consistent with the focus of the “challenge” from Dr. Puget alluded to above. Constraint programming has a major “declarative” aspect, in that a problem model can be handed off for solution to a variety of standard solving methods. These methods are embedded in algorithms, libraries, or specialized constraint programming languages. To fully exploit this declarative opportunity however, we must provide more assistance and automation in the modeling process.

In this paper we present our work in supporting automation in the modeling process through the use of natural sketch-based interfaces. While constraint programmers often draw constraint networks during the modelling process, there is no formalized method or language for drawing these diagrams. In this paper we choose an initial vocabulary based on the experience of the authors; part of this ongoing work includes determining the natural shape vocabulary for this domain. We take the view that in many interesting application domains, there is a natural graphical representation of the problem a user might wish to model. We present a sketch tool that can acquire a model of a constraint problem specified as a graph in which nodes are CSP variables and edges between variables represent constraints. The set of possible values each variable can take is specified by writing a set adjacent to a node. The constraint between two variables is specified by giving a direction on an edge and a relation adjacent to the edge. Our objective is to develop sketch recognition technology that can recognize hand-drawn representations of problems, and automatically generate constraint satisfaction models of them.

This paper describes a sketch recognition system that recognizes and solves a simplified set of hand-drawn constraint problems. Shapes are recognized using a combination of geometric and contextual rules, allowing shapes to be drawn freely, without requiring a specific drawing style.

2. Constraint Satisfaction and Modeling

Constraint satisfaction has become an important paradigm in artificial intelligence over the last 30 years [RvBW06, Mac77, Fre96]. Informally, a constraint satisfaction problem (CSP) is defined by a set of variables, each of which has a corresponding set of possible values called its domain, and the task is to find values for each variable so that a set of constraints is satisfied. Constraint satisfaction is applicable to a wide variety of problems arising in scheduling, design, configuration, machine vision, temporal reasoning and planning [Wal96].

To solve a problem with constraint satisfaction techniques one must first model the problem by a set of constraints that

every solution must satisfy. Unfortunately, considerable expertise is required to model a problem a set of constraints. Furthermore, many alternative models usually exist for a given problem, and expertise is required to formulate one of the more effective ones. Consequently, constraint technology is used only by enterprises that are large enough to have employees with this expertise. Researchers in the area of constraint programming have focused on addressing this so-called “modelling bottleneck” for many years [Smi06]. It is widely accepted the effective modelling is one of the major frontiers facing constraint satisfaction research [Fre99].

While most research on constraint modelling has focused on specification languages, problem reformulation, and expressive modelling languages, some researchers have developed icon-based languages for constructing constraint satisfaction models [Sim95, BB92]. In these systems users introduce new variables by introducing new icons onto a canvas and defining how the ports on these icons connect to other variables using constraints. These systems are quite usable, but we argue that they do not allow as natural a mode of interaction as sketching provides. It is for these reasons that we focus on sketching in the remainder of this paper.

3. Sketch Recognition Motivation and Goals

Sketched diagrams are important to the creative and learning processes; however, sketched diagrams take a long time for instructors to correct. And despite many studies saying that testing is extremely beneficial to the learning process [RK02], instructors will often omit student sketched diagrams from tests, and include more questions that can be quickly corrected. Sketch recognition systems, that automatically recognize hand sketched diagrams, can help bridge this gap, providing the creative visualization of a freeform piece of paper and the immediate feedback and functional visualization of a CAD system. Sketch recognition provides the following benefits: 1) encourage creativity through freeform drawing; 2) provide for active learning by making the student the creator; 3) enhance student learning through animated CAD diagrams; 4) provide immediate student feedback; 5) reduce teacher time by automatically correcting homework and test questions; 6) enhance and speedup teacher feedback by immediate feedback and collation of automatically corrected student grades.

This paper describes a sketch recognition system that recognizes a restricted set of constraint satisfaction problems. This system uses geometric-based rather than drawing style-based recognizers to allow freeform drawing. This is important since, firstly, the nature of the domain is one in which users will be brainstorming and not necessarily draw items in the same way each time, and secondly, we hope for wide usage in the classroom, and we want the recognition to be robust without requiring users to train the system with their drawing style, nor requiring the users to learn how to draw shapes in the system. Users should be able to use the system with no more training than would be necessary to sketch on a piece of paper.

Our goal is to allow users to draw without drawing style constraints, recognizing shapes by what they look like, rather than how they were drawn. To this end, we have the following goals for our recognition algorithm. Provided that shapes look like what is intended, they should be recognizable when:

1. drawn at any speed.
2. drawn in any order.
3. drawn in an interleaved manner, e.g., starting a second or third shape before finishing previous shapes.
4. strokes composing the shape are drawn in any order.
5. the number of strokes composing a shape is variable.
6. a single stroke may contain more than one shape or shape component.
7. drawn using any stroke direction.

To accomplish this goal, recognition is performed by decomposing strokes into their primitive components and then combining them hierarchically using geometric and contextual rules.

4. Previous Work in Sketch Recognition

Pen-input sketching and devices have been around since 1963, beginning with Ivan Sutherland's work on SketchPad [Sut63]. Sutherland did not actually recognize diagrams, but instead used a series of buttons to signify when he was drawing a circle, square, etc.,. Since then many efforts have been made to reduce drawing constraints and modalities to allow for more free-form sketching. Graffiti-type techniques were used on the Palm Pilot to take advantage of required drawing style, order, and direction to recognize hand-drawn shapes without requiring modal buttons to aid in recognition [Rub91, Lon01]. The disadvantage of such techniques is that they require users to learn how to draw objects in order to ensure they are correctly recognized. Vision techniques have been examined to recognize low-level objects, recognizing objects by what they look like rather than how they are drawn [GKSS05, Olt07], and while they offer considerably more drawing flexibility, this flexibility is limited to freedom in stroke number, order, and direction, as well as global scale and rotation; allowable local variations are not specifiable using these techniques. Geometric-based recognizers also attempt to recognize shapes by what they look like rather than how they are drawn, but instead, they parse strokes into geometric primitives [YC03, PH07] through corner detection [Her76, SSD01, KK06]. These primitives can then be combined hierarchically using constraints to specify the required geometric relationship and their allowable variations [Gro96, Ham07, AD04].

5. Sketch Recognition Methodology

The sketch recognition system described in this paper uses the LADDER/GUILD technology [Ham07] to produce the sketch recognition user interface. Shapes are described using the LADDER (LADDER = a sketch language to describe how shapes are drawn, displayed, and edited for use

in recognition) sketch recognition language in which shapes are described in terms of their primitives, the constraints that hold between them and other contextual shapes on the screen. Sketches are drawn on a TabletPC with the drawings saved as a series of strokes consisting of a series of points with (x,y,time) values. The GUILD (GUILD = generator of user interfaces from LADDER descriptions) system will automatically generate a user interface from these descriptions. GUILD parses strokes into primitives [PH07, SSD01], and joins them into higher-level primitives hierarchically using the rules given in the LADDER shape descriptions (these geometric and contextual rules would usually be called *constraints* that need to be satisfied to a certain threshold, but for the purposes of this paper, they will be called rules in order to avoid confusion with the sketch recognition domain being tackled here). Shapes can be edited and are displayed as specified by the LADDER shape descriptions as well. An interface supplied by GUILD allows a back-end to know what shapes are recognized to effectively process the diagram.

The system currently recognizes free-form constraint network diagrams that consist of labeled nodes that represent variables, labeled links between the nodes that represent constraints, and domain information about the allowable range of the variables. Text can be entered in three different ways: firstly, by tapping the screen on the diagram where the text should be placed, and then handwriting the text through the TabletPC; secondly, by tapping the screen on the diagram where the text should be placed, and then typing the text with the keyboard; or thirdly, a limited number of letters (t, v, w, x, y, z) can be handwritten directly on the diagram for use as variable names. Figure 1 shows the list of recognizable shapes in the domain. Shape recognition is performed such that shapes can be drawn in any order, using any number of strokes, and with strokes drawn in any direction.

5.1. Recognition Difficulties

When looking at geometrical properties alone, many shapes appear to be similar. For instance, the "less than constraint" (<), the V, the W, and the arrowhead all consist of the same general shape. It is only in context that we can distinguish these four different shapes. The following subsections describe the rules used to effectively recognize the different shapes. Note that many additional geometric or contextual rules follow by transitivity, but these are not included in the rules for simplicity.

5.2. Node Recognition

Nodes are recognized as ellipses that contain a variable name within them. Variables are either drawn in using the limited letters described below, or typed in, with no restriction on the variable name. LADDER provides for abstract shapes. All drawn and typed variables are of the abstract shape *Variable*. Arbitrary labels given to the lines and parts of shapes are arbitrary and for ease of explanation.

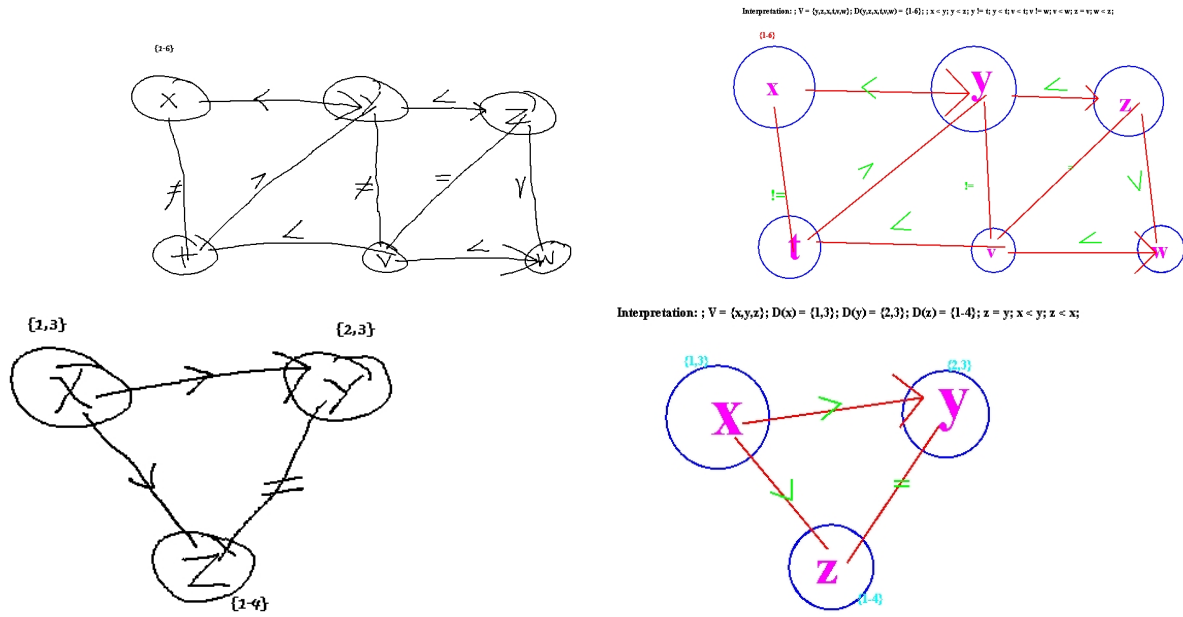


Figure 1: Two original hand-sketches successfully recognized by our system. The computer labeled recognition results by coloring nodes in blue, variables in magenta, constraints in green, and links and generalized domain text in red. The interpretation is shown at the top, with the variables listed first, the domain range listed second, and the list of specified constraints listed third. The interpretations are: (Interpretation: $V = \{y,z,x,t,v,w\}$; $D\{y,z,x,t,v,w\} = \{1,6\}$; $x < y$; $y < z$; $y = t$; $y < t$; $v \neq w$; $v < w$; $z = v$; $w < z$;) and (Interpretation: $V = \{x,y,z\}$; $D\{x\} = \{1,3\}$; $D\{y\} = \{2,3\}$; $D\{z\} = \{1-4\}$; $z = y$; $x < y$; $z < x$;) Each recognized item has also been replaced by a beautified version of the recognized text or shape.

5.3. Undirected Link Recognition

An undirected link is recognized geometrically as a line. extends between two nodes with the following rules.

1. The "start" node contains the endpoint labeled "p1".
2. The "end" node contains the endpoint labeled "p2".
3. The "start" node does not contain the endpoint labeled "p2".
4. The "end" node does not contain the endpoint labeled "p1".

5.4. Directed Link Recognition

Directed links are recognized as an arrow that extends between two nodes. The arrow is first recognized as three lines that meet at a point with the following geometrical rules:

1. The endpoint "p1" of line "shaft" is connected to the endpoint p1 of line "head1"
2. The endpoint "p1" of line "shaft" is connected to the endpoint p1 of line "head2"
3. The line "shaft" is longer than the line "head1".
4. The lines "head1" and "head2" are of equal length.
5. The lines "shaft" and "head1" meet at their endpoints and form an acute angle when traveling from line "shaft" to line "head1".
6. The lines "head2" and "shaft" meet at their endpoints and

form an acute angle when traveling from line "head2" to line "shaft".

Contextual rules are then used to recognize the arrow as a directed link:

1. The "start" node contains the "tail" of the arrow (defined as point "p2" of line "shaft").
2. The "end" node contains the "head" of the arrow (defined as point "p1" of line "shaft").

In the recognition process, a directed link may begin as an undirected link (when the two nodes are present), or as an arrow (when the nodes are not yet present until after the arrow is drawn). The recognizer handles this by keeping multiple interpretations at all steps along the way and modifying them as more information is present. (Described in more detail in further sections.)

5.5. Variable Letters

Variable letters are expected to be in their canonical orientation, as people rarely write an upside-down or slanted letters to identify a node. Thus we use this information to help distinguish and identify shapes. The variable labels are expected to be inside the node to help reduce ambiguity, but we expect this rule will be relaxed in the future. Recognition of

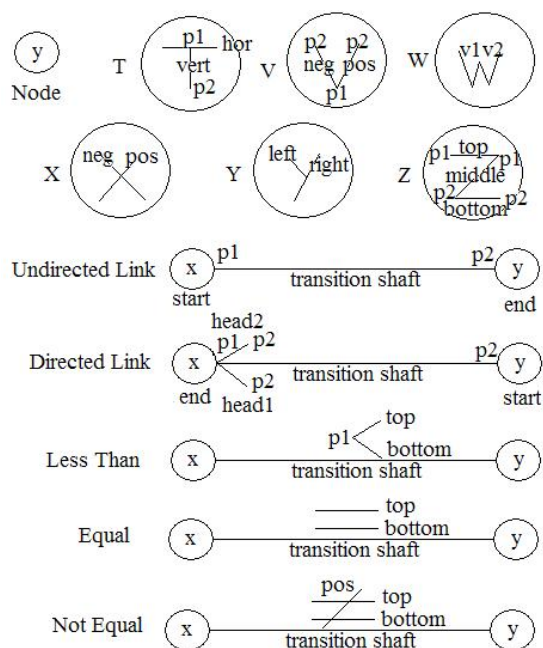


Figure 2: The recognizable shapes with the labels used in the recognition descriptions.

these letters is done through geometric and contextual features. In the future we would like to combine geometrical recognition with results from a handwriting recognizer to reduce the variable limitations and allow any text to be drawn. However, results from handwriting recognition system alone will not suffice as these results are highly dependent on sentence/grammar/dictionary word context which does not exist in variable terms for a constraint network. Thus, future improvements for recognizing text will involve integrating handwriting recognition results with diagram geometry and context to produce the highest recognition results. Future work also includes experimentation to determine if users would prefer to have variables automatically added to each node as it is drawn in an x_1, x_2, x_3, \dots fashion.

5.5.1. Variable Letter T Recognition

The letter T consists of two meeting lines abiding by the following geometric rules:

1. The "hor" line is horizontal.
2. The "vert" line is vertical.
3. The endpoint "p1" of the "vert" line bisects the "hor" line.
4. The center of the "hor" line is above the endpoint "p2" of the "vert" line.
5. The endpoint "p1" of the "vert" line is above the endpoint "p2" of the "vert" line.
6. The endpoint "p1" of the "vert" line does not bisect the "hor" line at its endpoint "p2".

Contextual rules are then used to further identify T:

1. The "node" contains the center of the "vert" line.

5.5.2. Variable Letter V Recognition

The letter V consists of two connecting lines abiding by the following geometric rules:

1. The "neg" line is negatively or vertically sloped.
2. The "pos" line is positively or vertically sloped.
3. The endpoint "p1" of "pos" line and the endpoint "p1" of the "neg" line are connected.
4. The endpoint "p2" of "pos" line is above the endpoint "p1" of the "pos" line.
5. The endpoint "p2" of "neg" line is above the endpoint "p1" of the "neg" line.
6. The endpoint "p2" of "neg" line is left of the endpoint "p2" of the "pos" line.
7. The "pos" line and the "neg" line are of equal length.

Contextual rules are then used to further identify V:

1. The "node" contains the center of the "pos" line.
2. The "node" contains the center of the "neg" line.

5.5.3. Variable Letter W Recognition

The letter W consists of two connected Vs. The geometric rules only require that two Vs be found. Contextual rules require that the two V's are collocated in the same node.

5.5.4. Variable Letter X Recognition

The letter X consists of two intersecting lines abiding by the following geometric rules:

1. The "neg" line is negatively sloped.
2. The "pos" line is positively sloped.
3. The "neg" line and "pos" line are of equal length.
4. The "neg" line and the "pos" line bisect each other.

Contextual rules are then used to further identify X:

1. The "node" contains the center of the "pos" line.

5.5.5. Variable Letter Y Recognition

The letter Y consists of two meeting lines abiding by the following geometric rules:

1. The "left" line has a negative slope.
2. The "right" line has a positive slope.
3. The endpoint "p2" of the "left" line bisects the "right" line.
4. The "right" line is larger than the "left" line.

Contextual rules are then used to further identify Y:

1. The "node" contains the center of the "right" line.

5.5.6. Variable Letter Z Recognition

The letter Z consists of three connecting lines abiding by the following geometric rules.

1. The "top" line is horizontal.
2. The "bottom" line is horizontal.
3. The "middle" line has a positive slope.

4. The endpoint "p2" of the "top" line is connected to the endpoint "p1" of the "middle" line.
5. The endpoint "p2" of the "middle" line is connected to the endpoint "p1" of the "bottom" line.

Contextual rules are then used to further identify Z:

1. The "node" contains the center of the "middle" line.

5.6. Constraint Recognition

The canonical orientation of a "less than" constraint is two lines connected at a single endpoint, pointing to the left. Another common orientation is its inverse, pointing to the right, forming a "greater than" constraint. However, we have found that other orientations are quite popular in this domain, as people will often sketch along the direction of the arrow, implying that the variable pointed at is less than the other.

5.6.1. Constraint "Less Than" Recognition

The "less than" constraint consists of two lines meeting at a point abiding by the following geometric rules.

1. The endpoint "p1" of the "top" line is connected to the endpoint "p1" of the "bottom" line.
2. The "top" line and the "bottom" are not parallel nor obtuse.
3. The "top" line and the "bottom" line are of equal length.

Contextual rules are then used to further identify the "less than" constraint:

1. The center of the "bottom" line is near the center of the transition.
2. The "shaft" of the transition is longer than the "bottom" line.
3. The center of the "bottom" line is near the center of the transition.
4. The center of the "bottom" line is closer to the center of the transition than to the "head" of the transition.
5. The center of the "bottom" line is closer to the center of the transition than to the "tail" of the transition.

5.6.2. Constraint "Equal" Recognition

The "equal" constraint consists of two parallel lines abiding by the following geometric rules:

1. The "top" line and the "bottom" line are of equal length.
2. The "top" line and the "bottom" line are parallel.
3. The center of the "top" line and the "center" of the bottom line are near to each other.

Contextual rules are then used to further identify the "equal" constraint:

1. The center of the "bottom" line is near the center of the transition.
2. The "shaft" of the transition is longer than the "bottom" line.
3. The center of the "bottom" line is closer to the center of the transition than to the "head" of the transition.
4. The center of the "bottom" line is closer to the center of the transition than to the "tail" of the transition.

5.6.3. Constraint "Not Equal" Recognition

The "not equal" constraint consisted of a line intersecting an already identified equal constraint, abiding by the following geometric rules:

1. The "pos" line intersects the "bottom" line of the "equal" constraint.
2. The "pos" line intersects the "top" line of the "equal" constraint.

Contextual rules are then used to further identify the "equal" constraint:

1. The "shaft" of the transition (from the "equal" constraint) is longer than the "pos" line.

5.7. Multiple Recognition Possibilities

The recognition descriptions above rely on endpoint ("p1" and "p2") labels requiring the line to be in a particular direction. At first glance this would imply that the system would have to know something about the drawing order to solve this. This is not the case. Rather, for every line recognized by the stroke parser, we add two lines, one in each direction. These lines are associated by a sort of shape-internal truth maintenance system. Shapes that are composed of the same subcomponents (such as two lines made from the same sub-stroke, or two different arrows made using the same line) are considered partners. In order for the recognition system to provide free-sketching ability, the system must keep all possible interpretations (or partners) in its database of possible shapes to compose higher-level shapes from. However, only one such interpretation can be chosen for display or for a final interpretation at any such time. In order to ensure that no subcomponents are shared in the final interpretation of the shape, each shape keeps track of its partners.

5.8. Fast Recognition

Keeping all possible interpretations can quickly become an intractable problem. To keep recognition response times short, as soon as shapes are recognized, their properties (such as x and y location, orientation, and width, height, and diagonal length of the total shape and subcomponents) are computed and placed into a hash-table. This hash-table provides ease of reference when forming hierarchical interpretations later and allows for quick pruning of the myriad of possibilities available for subshape combination.

5.9. Perception-Based Thresholding

Each of the geometrical and contextual rules to be held for the shapes in this domain have allowable thresholds which determine their truth value. Hand-drawn shapes always have a certain amount of noise to them, and determining these thresholds can be difficult. To give an example, distinguishing between the x and y letters is difficult and the two rules that distinguish them is the relative line length and that the endpoint of the negatively sloped line of the y bisects the positively sloped line, whereas the positive and negatively

sloped lines of the x bisect each other. Distinguishing between bisection and connected endpoints requires careful thresholding, as both rules are essentially just syntactic sugar for determining if two points are coincident.

A rational solution is to provide a single pixel-based threshold, such as requiring that intentionally coincident points be within 10 pixels (or some other logically chosen number) of each other. While this works in some cases, any solely pixel-based threshold will run into instances where it will either be too strict for large shapes drawn with long lines and/or too loose for short lines, causing otherwise perceptually connected lines not to be connected when drawn at a larger scale. A length-based threshold relieves these problems slightly, but it can cause the reverse problem, creating too strict thresholds for very small lines, and too loose for long lines. Combining the absolute and relative thresholds to give pixel-based upper and lower bounds helps to relieve this problem. However, in the case of small lines it can still be either too strict or may cause both endpoints to appear coincident, and not properly distinguishing between the different endpoints. Thus an additional perceptual property is required for coincident points that are line endpoints: if line endpoint "p1" is to be coincident to another point "p", then point "p" must be closer to endpoint "p1" than to the midpoint (else it would be perceptually appear to bisect the line). Likewise, if a point or line is said to bisect a line, the intersection point must be closer to the midpoint of the line bisected line than to either of its endpoints. The combination of these absolute, relative, and perceptual rules into a single error threshold gives a more accurate representation of the error associated with a particular geometric or contextual rule. Likewise, other geometric and contextual rules (such as near) are handled in a similar manner, merging absolute, relative, and perceptual rules to improve the threshold. The determination of these thresholds is currently a combination of values determined by the intuition of the developer along with those values empirically determined. Ongoing work includes updating all thresholds to be empirically determined.

5.10. Editing

The following editing behaviors are included:

1. Shapes in the domain can be deleted by scribbling them out.
2. Shapes can be moved by holding the pen over the center of the object and then dragging the object to the desired location.
3. Directed and undirected links can be rubber-banded, holding the pen over one end of the link, and then dragging it around to that that end remains attached to the pen and the other end remains fixed to the screen.
4. Nodes can be scaled by holding the pen at the bottom right corner and dragging the corner to the desired location.

6. Preliminary Results

This paper describes a work-in-progress, and as such, we have not yet fully tested our system on our target population.

As a preliminary proof-of-concept, we report the results of nine drawings created by members of the Sketch Recognition Lab at Texas A&M University. It should be noted that these users are familiar with sketch recognition technologies and their limitations, so these results are not meant to serve as a proof of a successful implementation that is ready for prime-time use. Rather, we present these informal results to show that the system can successfully recognize drawings. In our constraint network sketch recognition system, diagrams are recognized in real-time, with feedback presented to the user about the system's current understanding of the sketch, allowing users to easily correct their incorrectly parsed diagrams. In order to obtain initial recognition results, the drawings were collected on a separate input panel that performed no recognition and provided no feedback to the user. Results from the nine drawings obtained are shown in Table 1. Typed text, or text recognized through the TabletPC input panel, was not included in the recognition results, since it does not use the recognition results described here.

Screenshots of our system in action are shown in Figure 1. Although the hand-drawn figures were perfectly recognized (100% accuracy), these additional diagrams were not part of our study results described above because: 1) the diagrams were drawn using our sketch recognition user interface where feedback was provided if shapes were incorrectly recognized, and 2) they were drawn by the designers of this system. The GUILD system supplies an interface for accessing the recognized shapes. In order to emphasize the understanding of the hand-drawn constraints, a back-end system was written which contains only a for loop to print out each of the constraints specified in the system. A video showing our system in action is also available at: <http://srl.csdl.tamu.edu/projects/csp.html>.

7. Conclusions and Future Work

In this paper we have described a work-in-progress, and we have plans for many improvements in the near future. Immediate improvements include collecting natural hand sketches from target users, making the recognition robust (with high accuracy) for these users, improved beautification, integration into a sophisticated constraint satisfaction system, and extending the types of recognizable drawings to those of other popular constraint satisfaction problems, such as the four queens problem, map coloring, Sudoku, and satisfiability. The current system recognizes a small set of constraint satisfaction problems. Preliminary results show promise.

Acknowledgements. Tracy Hammond is supported by the National Science Foundation (Grant Number 0744150). Barry O'Sullivan is supported by Science Foundation Ireland (Grant Number 05/IN/I886). We thank Hadrien Cambazard (4C) and Brandon Paulson (SRL) for their comments.

References

- [AD04] ALVARADO C., DAVIS R.: Sketchread: A multi-domain sketch recognition engine. In *Proceedings of UIST '04* (2004), pp. 23–32.

Table 1: The recognition results from our preliminary evaluation of the system.

| trial # | nodes | directed links | undirected links | x | y | z | t | v | w | < | = | != | total |
|---------|-------|----------------|------------------|-------|------|------|-------|------|------|-------|------|------|---------|
| 1 | 3/3 | 1/1 | 2/2 | 1/1 | 1/1 | 1/1 | 0/0 | 0/0 | 0/0 | 2/2 | 1/1 | 0/0 | 12/12 |
| 2 | 3/3 | 1/2 | 1/1 | 1/1 | 1/1 | 1/1 | 0/0 | 0/0 | 0/0 | 2/2 | 1/1 | 0/0 | 11/12 |
| 3 | 3/3 | 2/2 | 1/1 | 0/0 | 0/0 | 1/1 | 0/0 | 1/1 | 1/1 | 2/2 | 1/1 | 0/0 | 12/12 |
| 4 | 3/3 | 2/2 | 1/1 | 0/0 | 0/0 | 0/0 | 1/1 | 1/1 | 1/1 | 0/0 | 2/2 | 1/1 | 12/12 |
| 5 | 4/4 | 3/3 | 2/2 | 1/1 | 1/1 | 1/1 | 1/1 | 0/0 | 0/0 | 2/2 | 2/2 | 1/1 | 18/18 |
| 6 | 4/4 | 3/3 | 2/2 | 1/1 | 1/1 | 0/0 | 0/0 | 1/1 | 1/1 | 3/3 | 1/1 | 1/1 | 18/18 |
| 7 | 4/4 | 2/3 | 2/2 | 0/0 | 0/0 | 1/1 | 1/1 | 1/1 | 1/1 | 2/2 | 1/2 | 1/2 | 16/19 |
| 8 | 5/5 | 2/2 | 2/2 | 1/1 | 1/1 | 0/1 | 1/1 | 1/1 | 0/0 | 3/4 | 0/0 | 0/1 | 16/19 |
| 9 | 6/6 | 5/5 | 3/3 | 1/1 | 0/1 | 1/1 | 1/1 | 0/1 | 0/1 | 4/4 | 0/1 | 1/2 | 22/27 |
| total | 35/35 | 21/23 | 16/16 | 6/6 | 5/6 | 6/7 | 5/5 | 5/6 | 4/5 | 20/21 | 9/11 | 5/8 | 137/149 |
| total % | 100.0 | 91.3 | 100.0 | 100.0 | 83.3 | 85.7 | 100.0 | 83.3 | 80.0 | 95.2 | 81.8 | 62.5 | 91.9 |

- [BB92] BOWEN J., BAHLER D.: Frames, quantification, perspectives, and negotiation in constraint networks for life-cycle engineering. *AI in Engineering* 7, 4 (1992), 199–226.
- [Fre96] FREUDER E. C.: In pursuit of the holy grail. *ACM Comput. Surv.* 28, 4es (1996), 63.
- [Fre99] FREUDER E. C.: Modeling: The final frontier. In *Proceedings of Practical Application of Constraint Technologies and Logic Programming* (1999).
- [GKSS05] GENNARI L., KARA L., STAHOVICH T., SHIMADA K.: Combining geometry and domain knowledge to interpret hand-drawn diagrams. *Computers & Graphics, Elsevier* (2005).
- [Gro96] GROSS M. D.: The electronic cocktail napkin - a computational environment for working with design diagrams. *Design Studies* 17 (1996), 53–69.
- [Ham07] HAMMOND T.: *LADDER: A Perceptually-based Language to Simplify Sketch Recognition User Interface Development*. PhD thesis, MIT, 2007.
- [Her76] HEROT C.: Graphical input through machine recognition of sketches. *Proceedings of the 3rd annual conference on Computer graphics and interactive techniques* (1976), 97 – 102.
- [KK06] KIM D. H., KIM M.-J.: A curvature estimation for pen input segmentation in sketch-based modeling. *Computer-Aided Design, Elsevier* (2006).
- [Lon01] LONG A. C.: *Quill: a Gesture Design Tool for Pen-based User Interfaces*. Eecs department, U.C. Berkeley, Berkeley, California, December 2001.
- [Mac77] MACKWORTH A. K.: Consistency in networks of relations. *Artif. Intell.* 8, 1 (1977), 99–118.
- [Olt07] OLTMANS M.: *Envisioning Sketch Recognition: A Local Feature Based Approach to Recognizing Informal Sketches*. PhD thesis, MIT, 2007.
- [PH07] PAULSON B., HAMMOND T.: A system for recognizing and beautifying low-level sketch shapes using ndde and dcr. In *ACM Symposium on User Interface Software and Technology (UIST2007)* (2007).
- [RK02] ROEDIGER H., KARPICKE J.: The power of testing memory: Basic research and implications for educational practice. *Association for Psychological Science* 1:3 (2002), 181–210.
- [Rub91] RUBINE D.: Specifying gestures by example. In *Computer Graphics* (1991), vol. 25(4), pp. 329–337.
- [RvBW06] ROSSI F., VAN BEEK P., WALSH T. (Eds.): *Handbook of Constraint Programming*. Elsevier, 2006.
- [Sim95] SIMONIS H.: Application development with the chip system. In *CDB* (1995), Kuper G. M., Wallace M., (Eds.), vol. 1034 of *LNC3*, Springer, pp. 1–21.
- [Smi06] SMITH B. M.: Modelling. In *Handbook of Constraint Programming*, Rossi F., van Beek P., Walsh T., (Eds.). Elsevier, 2006, ch. 14.
- [SSD01] SEZGIN T. M., STAHOVICH T., DAVIS R.: Sketch based interfaces: Early processing for sketch understanding. In *Proceedings of Perceptive User Interfaces Workshop* (November 2001).
- [Sut63] SUTHERLAND I. B.: Sketchpad, a man-machine graphical communication system. *Proceedings of the Spring Joint Computer Conference* (1963), 329–346.
- [Wal96] WALLACE M.: Practical applications of constraint programming. *Constraints* 1, 1/2 (1996), 139–168.
- [YC03] YU B., CAI S.: A domain-independent system for sketch recognition. In *Computer graphics and interactive techniques in Australasia and South East Asia* (2003).