EUROGRAPHICS Symposium on Sketch-Based Interfaces and Modeling (2010)
M. Alexa and E. Do (Editors)

# Sketch-Based Recognition System for General Articulated Skeletal Figures

S. Zamora[1] and T. Sherwood[1]

[1]Department of Computer Science, University of California, Santa Barbara

**Abstract**

*We present a new recognition system for detecting general articulated skeletal figures in sketch-based applications. We abstract drawing style from recognition by defining figures using two data models: templates and figure targets. Our system recognizes general skeletal figures consisting of lines and ellipses which may be drawn partially off the canvas with the system automatically completing the figure. Figures are modeled as graphs and are allowed to contain cycles. Subgraph matching on a graph built from the stroke input is used to perform recognition. This paper outlines our system design, key details to its proper implementation, and proposes its application to various domains of sketch recognition.*

Categories and Subject Descriptors (according to ACM CCS): I.4.8 [Image Processing and Computer Vision]: Scene Analysis—Object recognition

## 1. Introduction

Stick figures are a universal staple of sketching that effectively demonstrate the sort of expressiveness that motivates research into sketch recognition. Each figure conveys a great deal of information – an entity is represented (such as a human being) in a given position and orientation on the canvas. The strokes of a sketched figure outline a structure of bones and joints in a specific pose with each bone in its own position and orientation. The resulting composition is an articulated skeletal structure that we can define using a graph of connected strokes.

However, stick figures may be drawn in different styles; some may draw a torso with an ellipse while others may use a single line. They are also often drawn roughly; lines may not touch or could cross (undershoot and overshoot), cross junctions can be expressed using two, three, or four lines, etc. Figures may even be drawn partially offscreen, as often the case with human stick figures where only the torso is important and the rest of the body is 'off camera', so to speak. But even when drawn partially offscreen, these figures remain recognizable by a human observer. While sketch recognition systems strive to be robust under noisy input, robustness to drawing style and abstraction is a different class of problem.

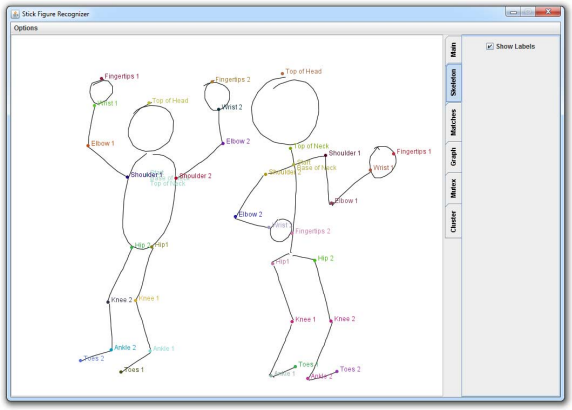


**Figure 1:** *Two human stick figures sketched and detected by our skeletal figure recognizer. Because the templates for the recognized figures identify both as 'Human', the two figures are annotated with labels from the 'Human' figure target.*

In order to design a stick figure recognizer to handle style and abstraction, we generalize our approach to the more general *skeletal figure*, a general articulated skeleton of lines and ellipses. In order to abstract from the application the style by which a figure is drawn, we separate drawing styles (modeled as *templates* which outline a general skeletal figure)



DOI: 10.2312/SBM/SBM10/119-126

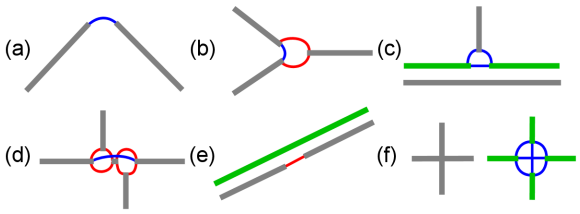


**Figure 2:** *The six graph elaboration strategies – (a) proximity linking, (b) link closure, (c) virtual junction splitting, (d) spurious segment jumping, (e) continuity tracing, and (f) crossed segment co-splitting. Grey/blue lines denote bonds/links present before the strategy is applied. Green/red lines denote new bodied/non-bodied links created by the strategy. In (f), the right structure replaces that on the left.*

from the class of figure (such as Human, Cat, Elephant, etc., modeled as *figure targets*) to be recognized. We then extend previous work in the field that has been shown to be very effective in recognizing skeletal figures [MF02b]. The contributions in our paper include a new recognition system for general articulated skeletal figures consisting of lines and ellipses, detection of partially drawn figures that extend off the canvas, and support of skeletal figures that include cycles.

## 2. Related Work

Hammond and Davis developed LADDER [HD05], a cross-domain framework for creating recognizers from descriptions written in the LADDER language. While many sketch applications have been developed using LADDER across a variety of domains [TH10] [PEW*08] [TPH09], its language is geared for diagrams with spatial (above, left of, etc.) and angular (acute, vertical, etc.) constraints and is not suited for general articulated figures that could be drawn irregularly (e.g. a T junction drawn with two lines versus three). Examples of stick figures in LADDER can be found using such constraints [HD03]. LADDER also does not support rotational constraints on ellipses as described in Section 3.

Alvarado and Davis developed SketchREAD [AD04], a recognition system similar to LADDER that also uses a template language to define recognition. SketchREAD also uses relative and angular constraints that make it unsuitable for detection of articulated skeletal figures. Both of these projects require templates to be hand-coded, although a graphical tool for finding bugs was developed for LADDER [HD06]. The Electronic Cocktail Napkin project [GD96] allows templates to be generated from sketched examples but only builds constraints from relative and loose spatial relations (concentric, contains, overlaps, etc.).

Lee et al. [LBKS07] developed a graph based recognizer that functions similarly to the model and data graph approach used in our system. Their recognizer uses offline graph matching with training data to build a template to recognize, and uses subgraph matching in order to recognize the stroke input. However, their matching algorithm takes into account relative angles between primitives, which would not support articulated figures. Sezgin and Davis [SD05] used Hidden Markov Models to build a recognizer that takes advantage of the consistency of an individual's stroke drawing order. While their examples and user studies involved recognizing stick figures, recognized lines are identified specifically as horizontal, vertical, and positively or negatively sloped, making the recognizer sensitive to rotation and unable to handle articulated figures. The recognizer used by Yang et al. [YSvdP05] also uses graph based template matching. Their system does not require figures to be fully connected and uses a hybrid spatial and structural approach for matching sketches to templates. However, their system does not fully discern the structure of the sketched scene and is not designed for articulated, rotationally-invariant figures.

Our work extends Mahoney and Fromherz's stick figure recognition system [MF02a] [MF02b] [MF01] which models stick figures as graphs that are matched to a graph generated from the stroke input. However, their implementation only recognizes a single figure consisting only of lines and does not have support for ellipses. Their implementation does not support partially offscreen figures, and does not explore the extension to generalized skeletal figures that effectively applies the recognizer to a variety of domains.

To overview the work we are extending, Mahoney first represents the templated figure to be recognized as the *model graph*, where bones of the figure are nodes and joints between bones are edges. The sketched input data is represented as the *data graph* where the bodies of lines are edges called *bonds* and their endpoints serve as nodes. Relations between strokes in the data graph are edges called *links*. A process called *graph elaboration* is used to add the spatial and visual structure of the sketched scene to the data graph. This process adds links, alternate line segments, and mutual exclusion constraints (to prevent a stroke and its alternate interpretations from participating in a match together) to the data graph. Figure 2(a–e) outlines the original elaboration strategies used in Mahoney's algorithm. These elaboration strategies also make the recognizer tolerant to noise in the sketched figure, including line overshoot and undershoot at junctions. These strategies are discussed in Section 4.1.

Finally, Mahoney's system performs subgraph matching to find the best match for the model graph in the data graph. Considerations of mutual exclusion constraints and components marked as optional are applied. The best match that maximizes the number of matched parts and minimizes an energy function supplied with the model graph (such as the one described in Section 5.3) is then returned. A key benefit of this approach is that new strokes only add structure to the data graph and do not destroy substructures that could produce matches.

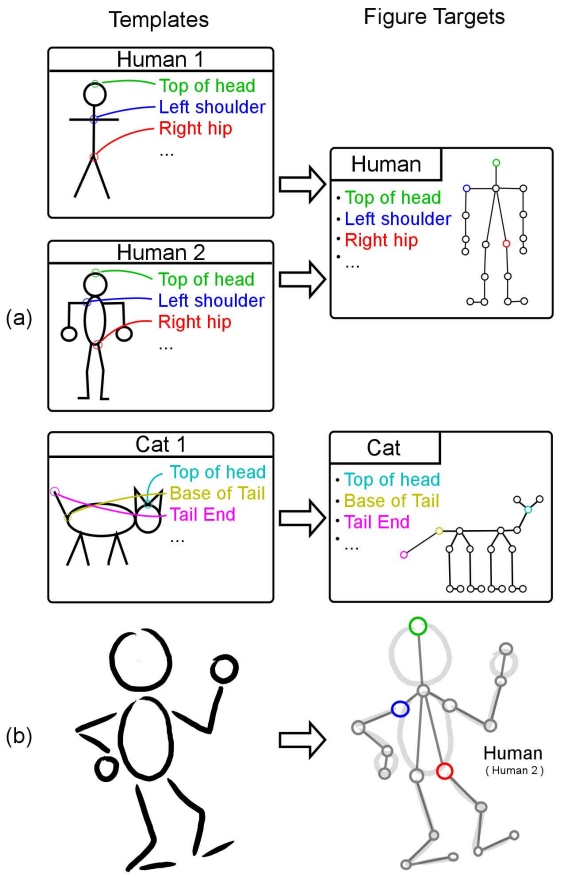


**Figure 3:** *Overview of our system for abstracting drawing style from figure class. In (a) we model the type of figure to recognize (Human, Cat, etc.) as a Figure Target and each style of drawing the figure target as a Template. When a templated figure is sketched, as shown in (b), the sketch is annotated with the skeletal structure of the figure target.*

## 3. System Design

Our key motivation lies in handling the variety of styles used for sketching skeletal figures. Even though human stick figures are drawn in a number of styles, with different configurations of lines and ellipses for heads, torsos, and limbs, all are recognizable by an observer as a 'human' figure. Our system mirrors this sort of polymorphism using two models: *templates* and *figure targets*. Figure targets outline a class of figure the system will match (e.g. Human, Cat, Arrow), while templates define each style of drawing the given figure, as illustrated in Figure 3(a). When recognition occurs in 3(b), the sketched figure that matches a given template is annotated with data points from the figure target. Our motivation is satisfied by abstracting from the application the styles by which the user can draw, for instance, a human stick figure, and exposing that they have simply drawn a human.

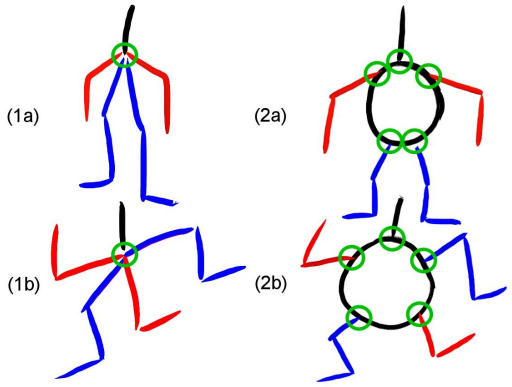


**Figure 4:** *This figure motivates rotational constraints on ellipses. (1a) and (1b) are a control and posed human with a point as a torso. (2a) and (2b) are a control and posed human with an ellipse as a torso. As junctions consisting of all lines collapse to a single point, ( shown by a single green circle in (1a) and (1b) ) and junctions involving ellipses do not ( shown by many green circles in (2a) and (2b) ), (2b) should not be recognized as its right leg and arm are swapped.*

The system performs the following steps in order to recognize sketched figures:

- Drawn strokes are classified as lines and ellipses and added to our data graph. Special care is given to strokes drawn near the edge of the canvas which can be identified as extending "offscreen" (Discussed in Section 4).
- After each stroke is drawn, graph elaboration is used to add connectivity relations and alternate interpretations of the sketched scene's structure to the data graph. These alternate interpretations are kept seperate using mutual exclusion constraints so no two interpretations of the same set of strokes can be used in the same match.
- Each template loaded into the recognizer (each used as a separate model graph) is checked against the data graph using subgraph matching. If an instance of the template is found in the data graph, it is added as a candidate figure.
- A fitness metric for each candidate is used to sort the set of candidate figures, and the set of best fitting, distinct candidates (not sharing any line segments or ellipses) are chosen as the set of recognized figures in the scene.
- The strokes for each recognized figure are annotated with the set of key points defined by it's template's figure target. Applications that work only with these key points can then ignore the particular template (i.e. drawing style) used to recognize the figure.

Figure targets define these key points as *anchor points*. A human figure target may have anchor points such as 'Top of Head', 'Shoulder 1', 'Elbow 1', etc. Each figure targets is a graph where anchor points serve as nodes. These nodes are connected in a way that is meaningful to the figure's skeleton – hips connect to knees, knees connect to ankles, etc. Anchor points may also be flagged as optional, identifying

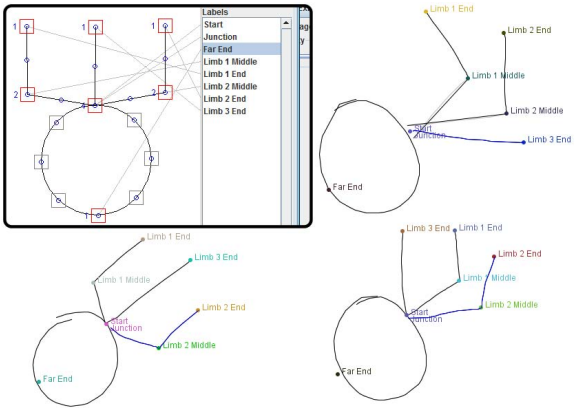


**Figure 5:** *Rotational constraints are not imposed within child groups of an ellipse. The template definition for a skeletal figure is given in the top left. In the remaining three figures we see three valid matches for the given template.*

components of a figure that may not present in all drawing styles, such as a human figure's hands and feet.

Templates serve as the model graphs used during subgraph matching. Templates are mapped to a given figure target by labeling the template's nodes with *anchor labels* from the figure target. When a sketched figure is matched against a template, these labels are used to annotate positions on the line segments and ellipses of the sketched figure. Since all templates that correspond to a figure target share the same label set, if an application only works with the set of the labels on detected figures, it will be able to ignore the style by which a figure has been drawn.

Our algorithm uses both lines and ellipses as *primitive* elements of the data and model graphs. We support ellipses by introducing *rotational constraints* that enforce a rotational ordering of components along the contour of an ellipse. As we have modeled ellipses as nodes in the model and data graph, these rotational constraints are necessitated by the distinctions between line endpoint and ellipse junctions as illustrated in Figure 4. But since some figures may not require rotational constraints between all of their components (exemplified in Figure 5) we organize primitives connected to an ellipse in the model graph into *n child groups* and impose rotational constraints between (but not within) these groups. These constraints are necessary but significantly complicate the structure of the graphs as well as the matching process.

There are five node types in our model graph: *line*, *ellipse*, *anchor*, *reference*, and '*node*'. Line and ellipse nodes represent the line and ellipse primitives the model graph will match in the sketched scene. The connectivity of the skeletal figure to be matched is mirrored by the edge connections between these primitive nodes in the model graph. Ellipses also define 8 child groups in counter-clockwise order at 45 degree angles across the ellipse contour. As illustrated in Figure 6,

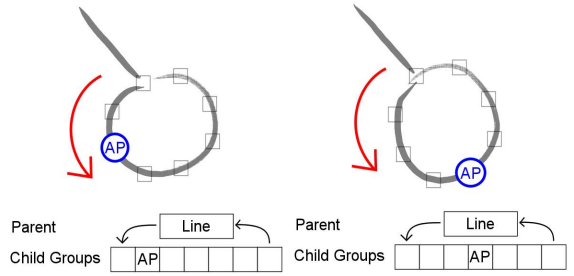


**Figure 6:** *The placement of leaf anchor nodes in an ellipse's set of child groups determines the position of the anchor point on matched figures.*

this allows for anchor labels to be positioned at different locations on the ellipse using anchor nodes.

Anchor nodes are placed in the model graph at joints between a set of connected lines and ellipses. Labels applied to anchor nodes will then position the anchor point on a matched figure on the joint between the set of corresponding strokes. Along with anchor nodes, labels may also be applied to lines. Labels applied to lines will position the anchor point on a matched stroke's midpoint. This adds additional flexibility – a human stick figure using a single line to represent a leg can place a knee anchor point at the line's midpoint.

All templates use another label not inherited from their figure target called "Start". The node labeled start defines the start point on the template where subgraph matching will begin. Our model graph is then modeled as a *model graph tree* rooted at the node labeled start. If a line is labeled start its node will be the root of the model graph tree. If a joint is labeled start, then a '*node*' node is set as the root of the tree indicating that subgraph matching should begin at *clusters*, as defined in Section 5. An example template and corresponding model graph tree is illustrated in Figure 7(a) and 7(c). Note also that as ellipses will always have a parent in the model graph tree, only 7 of its child groups will be used for children. Lines at the root of the model graph tree have 2 child groups for both its endpoints.

Reference nodes are used to represent cycles in the model graph tree. This allows templates and the drawing styles they capture to include closed shapes. These reference nodes target an ellipse or line node elsewhere in the model graph tree. During subgraph matching a candidate figure must be able to satisfy the cycles denoted by reference nodes in order to be a valid candidate.

Finally, the arrangement of optional anchor labels may allow some components of a matched figure to be omitted. Determining which lines and ellipse nodes in the model graph are optional for a general skeletal figure requires caution as optional anchor labels can be placed anywhere in the template. We identify nodes in three ways - *non-optional*, *partially optional*, and *pure optional*. Non-optional nodes must be matched, while pure optional nodes may be absent. Par-

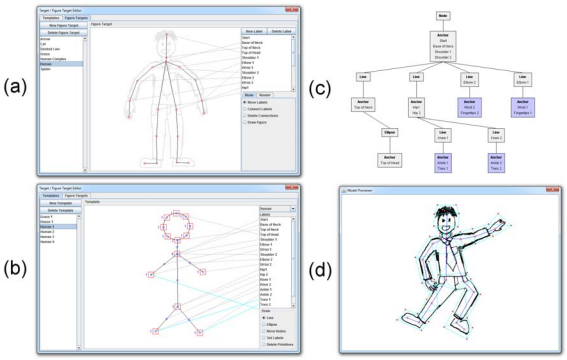


**Figure 7:** *Our editor for creating figure targets and templates. (a) shows a human figure target. (b) shows a human template and its matching model graph tree is shown in (c). (d) shows our skeletal animation previewer, allowing the user to drag around anchor points and deform the model.*

tially optional nodes which are assigned both optional and non-optional anchor labels are also required to be present for a match. Using this categorization, a recursive operation starting at the root node of the model graph tree can be used to determine the optionality of a templates line and ellipse components.

## 4. Generating the Data Graph

Strokes are categorized as lines, arcs, polylines, ellipses (including circles), or complex, using shape recognizers described by Paulson and Hammond [PH08] and a corner detection scheme described by Sezgin [SD04]. Lines and arcs are added to the data graph as two *line endpoint* nodes connected by a *bond* edge. Ellipses are added as an *ellipse* node. Polylines are decomposed and added as line segments. Complex strokes are treated like polylines but adjacent substrokes are connected with *direct* links as described in Section 4.1.

Graph elaboration may create *bodied links* in the data graph which function like bonds and can be matched during recognition. Bodied links are alternate interpretations of strokes in the sketched scene. For example, two adjacent collinear strokes are merged as a bodied link. Mutex constraints placed between the bodied link and its source components ensure that a stroke is not matched multiple times in a template under different interpretations.

To support partially offscreen figure detection we identify lines and ellipses that extend offscreen. If a line endpoint is within a threshold of the canvas edge we identify that endpoint and line as *variable extension*. If both endpoints of an arc are near the canvas edge we classify it as a *virtual ellipse*.

### 4.1. Elaboration Strategies

The graph elaboration strategies we employ are outlined in Figure 2. With the exception of CSC, these strategies are detailed in [MF02b]. Our system's utilization of these strategies differs from Mahoney's work in the following ways:

- We add a new elaboration strategy called *crossed segment co-splitting* (CSC). Because our recognizer is designed for tablet input (as opposed to a scanned document, as per Mahoney's approach), CSC is needed to decompose pairs of crossed strokes into four separate line segments. Four new line endpoint nodes are added at the intersection point and the original bonds are replaced with four new bonds between the new and original line endpoints. Direct links are placed between subsegments of the original bonds, and regular links are placed to fully connect the new endpoints. Lines are similarly split when they pass through ellipses.
- Proximity linking (PL) adds links between free ends, defined as line endpoints not connected to any non-bodied links. In order to handle cycles induced by sketched closed structures, we also allow PL to link any line endpoints with distances that lie below a threshold.
- Link closure (LC) does not need to be performed due to our explicit handling of *clusters* as defined in Section 5.
- While Mahoney only describes handling instances of virtual junction splitting (VJS) independently, we locate all endpoints that split a bond $B$. If $n$ free endpoints are found that create virtual junctions, $n+1$ bodied links are created between the $n$ free endpoints and the two endpoints of $B$. A mutex constraint is made between all the generated bodied links and $B$.

### 4.2. Applying the Strategies

Close consideration of graph elaboration finds that caution must be made in applying these strategies. Improper ordering will result in an mis-elaborated graph, which will not recognize figures that should be recognized. For example, if PL is applied before VJS, two lines drawn at a T junction will fail to create a virtual junction and instead connect the lines at their closest endpoints.

We apply an elaboration step each time a stroke is added to the scene. This is run once for lines and ellipses and once for each line segment in polylines and complex strokes (for complex strokes the direct links are created between the line segments after all of these elaboration steps. Elaboration is then run once more without adding any new primitives).

We begin an elaboration step by removing all of the bodied links, non-bodied links, and mutex constraints from the data graph, with the exception of links created during CSC. We then perform CSC using the added primitive, checking if an added line is to be split with other bonds or if an added ellipse should split existing bonds.

We next combine VJS, spurious segment jumping (SSJ), and what we call *round 1 continuity tracing* (R1CT). As VJS and SSJ can be easily combined into a single pass on a candidate line, this pass is augmented with continuity tracing. If

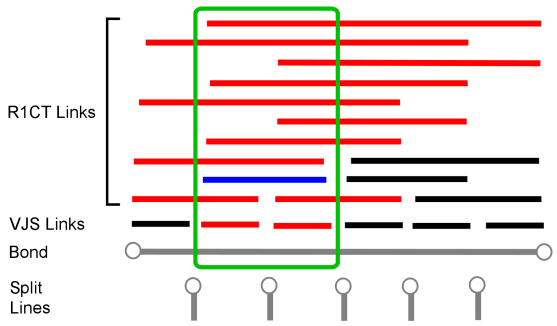


**Figure 8:** *Generated R1CT bodied links is shown in this figure. Six bodied links are created by VJS amongst which all contiguous subsets of lengths* $[2,5]$ *form R1CT bodied links. The blue R1CT bodied link is mutexed against all links within the green box ( highlighted in red ) and the bond.*

$B$ is split into $n$ bodied link segments, then $n(n-1)/2-1$ bodied links should be made to cover all sets of $i$ adjacent segments, where $i=[2,n-1]$, as illustrated in Figure 8. Mutex constraints (also highlighted in Figure 8) are carefully placed to enforce all appropriate constraints within the set of new bodied links and the original bond. We apply this VJS/SSJ/R1CT process to every bond in the data graph.

We then perform PL, followed by *round 2 continuity tracing* (R2CT) which functions as described in Section 4.1. R2CT is applied to all of the bonds and bodied links in the data graph, including those created during R2CT.

## 5. Subgraph Matching

After graph elaboration we recognize figures using subgraph matching. We initialize a candidate match for template $T$ at appropriate locations based on the root node of $T$'s model graph tree. Matching can begin at free endpoints, ellipses, and *clusters* - subgraphs in the data graph of non-bodied links and line endpoint nodes. For each candidate we walk through the data graph and assign primitives to nodes in $T$'s model graph tree with regard to mutex constraints. Anchor points for anchor nodes in the model graph tree are placed at the centroids of the set of line endpoints and ellipse connection points that participate in the joint. Anchor points for labels placed on lines are positioned at the midpoint between the line's endpoints. We also make sure that primitives assigned to reference nodes match the assignment to the targeted model graph node to support cycles.

Once the model graph tree has been satisfied, we have found a match. Our implementation finds all possible matches in the scene, sorts them by fitness, and returns the best distinct figures (that do not share strokes).

### 5.1. Enumerating All Possible Matches

At each step during the matching process, there exists a set $S$ of all lines and ellipses that can be used from our current position in the data graph. All permutations and combinations of $S$ that can satisfy the next group of primitives to be matched (with respect to rotational ordering along ellipses, mutex constraints, etc.) must be explored in order to find all possible matches. As shown in Figure 9, exploring all valid subsets of $S$ can result in factorial growth of the search space, but must be done to locate the user's intended figure.

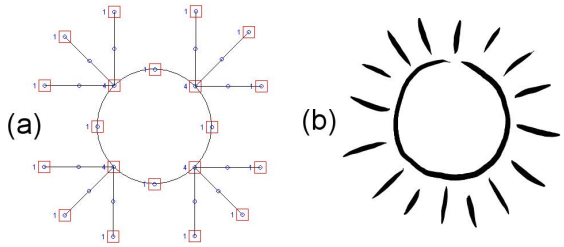


**Figure 9:** *Template and sketched figure exemplifying a factorial expansion of the search space. 28,304,640 possible matches exist between the sketch and the template.*

### 5.2. Partially Offscreen Figures / Optional Components

If variable extension lines are matched or optional components are not matched, the subtree of the template rooted at that element is ignored and effected anchor point locations are extrapolated using relative angles and edge distances in the figure target and the scaling factor of the drawn figure (as defined in Section 5.3). If a virtual ellipse is matched, we observe that any of the virtual ellipse's children in the model graph tree may be matched to an imaginary primitive on the contour of the virtual ellipse, requiring additional consideration to properly enumerate all possible assignments.

Note that because matching begins at the root of the template's model graph tree, the joint or line labeled start must be present in the canvas for an offscreen figure to be detected. This should be kept in mind when determining what element should be labeled start on a template.

### 5.3. Determining Fitness

We define fitness using the least squares error of each primitive's scale in the drawn figure against its matching element in the template. To support scale invariance, we use the matches *scaling factor*. Given $n$ elements that make up the template, $t_i$ gives the size of the element $i$ in the template (defined as the length of a line and diameter of the circle representing an ellipse), and $d_j$ gives the size of the primitive $j$ in the scene (the distance between endpoints for lines and twice the average distance of points along the stroke from the stroke centroid for ellipses), our error is computed as:

$$\text{scaling factor} = \frac{1}{n}\sum_{i=1}^{n}\frac{t_i}{d_i}$$

$$\text{error} = \sum_{i=1}^{n}(t_i - \text{scaling factor} * d_i)^2$$

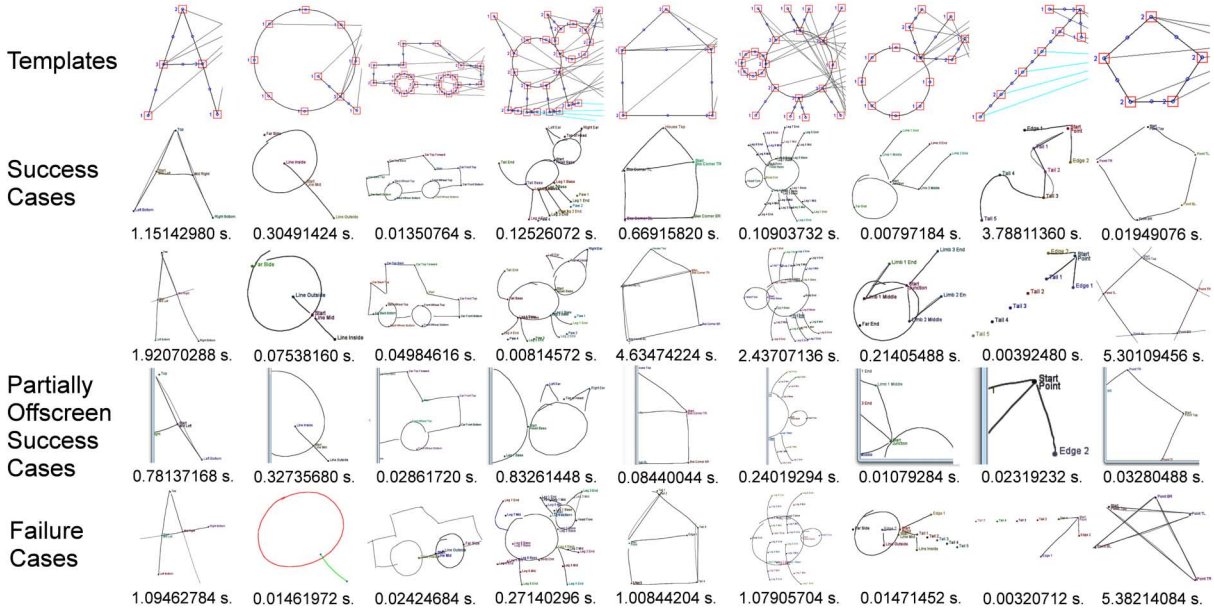


**Figure 10:** *Nine symbols from various domains modeled, identified, and annotated by our recognizer. The template for the symbol is given at the top, and below are 3 successful recognitions (the third being partially offscreen) and 1 failure case. These failures occur for different reasons, such as wrong primitives being matched (1,8), mis-elaboration (2), template mismatching (3,4,5,7), and illogically articulated yet positive results (6,9). Computation times are given for the combined elaboration and recognition step after the final stroke added to each figure. All 9 templates were loaded in the system during the matching phase for each trial. These times are only given for reference - our implementation is non-optimal.*

If $i$ is not present in the match due to being completely offscreen, $d_i$ is set to $t_i$. If $i$ is a variable extension line then $d_i$ is set to $max(d_i, t_i)$. If $i$ is a virtual ellipse then $d_i$ is set to the diameter of the circle found using the two endpoints and center point of the stroke of the virtual ellipse. A fitness score is determined using this error, the percentage of matched non-optional components, and the percentage of matched optional components.

## 6. Implementation

We implemented our skeletal figure recognizer in Java 1.6. Matched figures are annotated with anchor points, as seen in Figure 1. Our application also uses the anchor points to pose a skeletal mesh built on our figure targets using traditional skeletal animation methods. Java3D was used for texture mapping our posed skeletons.

Figure 7 shows our GUI-based editor for generating figure targets and templates. Figure targets are built by defining each anchor point and noting if they are optional. Anchor points are then dragged into place in the canvas and edges are made between them. When the graph is connected the figure target is marked as *ready*. To support skeletal animation and texture mapping, a paint brush is given to draw a caricature of the figure on the canvas. Vertices of the mesh are added to the canvas and manually triangulated by the user. An anchor point is chosen as the 'root' of the figure target's skeleton and a hierarchy of bones is generated. Vertices can be assigned to two bones with a weighting factor. A previewer, seen in Figure 7(d), allows the user to drag anchor points and preview how their skeleton deforms.

Building templates involves adding lines and ellipses to a canvas. Two nodes are created on the endpoints of each line, and eight nodes are created along the contour of each ellipse. Lines and ellipses are connected by being drawn so that their nodes merge. The figure target the user intends the template to reference is chosen from a drop-down list of ready figure targets. Choosing a figure target presents a list of its anchor labels. Once the user assigns each label to nodes and line midpoints, and if the graph of components is connected, the template is marked as ready. The optionality of primitives is automatically inferred based on the distribution of the anchor labels. All ready templates are then used in the main application's recognition step.

## 7. Conclusions

Our recognizer works as intended and has a good recognition rate of sketched symbols. Although the system is designed to recognize articulated figures, the generality of our approach and its support for templates that contain closed shapes allows the recognizer to accept a wide variety of symbols, including those which are not articulated. Any symbol that is connected and consists of lines and ellipses can be

found by our recognizer. Figure 10 shows our system recognizing a variety of symbols, from humans and animals to geometric shapes, houses, cars and arrows. Although rigidly defined symbols could be recognized by our system as inappropriately articulated, our system is easily extended with high level sanity checking to cull these matches.

Although our system can be used for a variety of general recognition tasks, its focus on articulated figures can be valuable to many domains with more specified needs. For example, artists who resort to motion capture or keyframing to create animations for characters could instead draw a series of stick figures from which an animation could be inferred (such as in [DAC*03]). In a storyboarding application, figures can be added and posed into storyboard cells quickly and naturally. The support for partially offscreen figures would allow artists to explore shot composition.

As for its limitations, our recognizer will not work on unconnected symbols (e.g. analog circuit symbols such as capacitors and batteries). Due to the rotational constraints placed on ellipses, mirroring across ellipses cannot be handled without making a second, mirrored template. Without sanity checking, inappropriate matches with superior fitness metrics may be favored by the recognizer. These can result from matching unintended primitives, wrong templates, mirroring (e.g. stick figures with left and right limbs may be detected as reversed), etc. Unless the elaboration steps are properly tuned with the right parameters, the data graph may be improperly elaborated such that that a valid sketch may not be recognized. Finally, our recognizer is a proof of concept and uses many brute force solvers that recompute large amounts of data each time a stroke is added. We present some failure cases and computation times in Figure 10 which could be improved upon by an optimal implementation, tuned parameters and fitness metrics, and sanity checking.

For further work, developing an implementation of our recognizer with an acceptable time performance is critical. Better heuristics for exploring the search space, determining optimal thresholds and parameters for our elaboration strategies, and holding a user study to assess our recognizer's effectiveness are important for preparing our recognizer for real world scenarios. We would also like to explore additional elaboration strategies such as *offscreen junctions*, where pairs of variable extension lines or virtual ellipses are extrapolated and checked for intersection points that lie offscreen. Finally, we would like to integrate the sorts of spatial constraints found in other recognition frameworks, with care taken to uphold our generality and focus on articulation.

## References


[AD04] ALVARADO C., DAVIS R.: Sketchread: a multi-domain sketch recognition engine. In *UIST '04: Proceedings of the 17th annual ACM symposium on User Interface Software and Technology* (2004), pp. 23–32. 2

[DAC*03] DAVIS J., AGRAWALA M., CHUANG E., POPOVIĆ Z., SALESIN D.: A sketching interface for articulated figure animation. In *Eurographics/SIGGRAPH Symposium on Computer Animation, SCA* (2003). 8

[GD96] GROSS M., DO E.: Demonstrating the electronic cocktail napkin: a paper-like interface for early design. In *CHI '96: Conference companion on Human factors in computing systems* (1996), pp. 5–6. 2

[HD03] HAMMOND T., DAVIS R.: Ladder: a language to describe drawing, display, and editing in sketch recognition. In *IJCAI'03: Proceedings of the 18th international joint conference on Artificial intelligence* (2003), pp. 461–467. 2

[HD05] HAMMOND T., DAVIS R.: Ladder, a sketching language for user interface developers. *Computers & Graphics 29*, 4 (2005), 518–532. 2

[HD06] HAMMOND T., DAVIS R.: Interactive learning of structural shape descriptions from automatically generated near-miss examples. In *IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces* (2006), pp. 210–217. 2

[LBKS07] LEE W., BURAK KARA L., STAHOVICH T. F.: An efficient graph-based recognizer for hand-drawn symbols. *Computers & Graphics 31*, 4 (2007), 554–567. 2

[MF01] MAHONEY J. V., FROMHERZ M. P. J.: Handling ambiguity in constraint-based recognition of stick figure sketches. In *Document Recognition and Retrieval IX* (2001), vol. 4670, pp. 89–100. 2

[MF02a] MAHONEY J. V., FROMHERZ M. P. J.: Interpreting sloppy stick figures by graph rectification and constraint-based matching. In *GREC '01: Selected Papers from the Fourth International Workshop on Graphics Recognition Algorithms and Applications* (2002), pp. 222–235. 2

[MF02b] MAHONEY J. V., FROMHERZ M. P. J.: Three main concerns in sketch recognition and an approach to addressing them. *In Sketch Understanding, Papers from the 2002 AAAI Spring Symposium* (2002), 105–112. 2, 5

[PEW*08] PAULSON B., EOFF B., WOLIN A., JOHNSTON J., HAMMOND T.: Sketch-based educational games: "drawing" kids away from traditional interfaces. In *IDC '08: Proceedings of the 7th international conference on Interaction design and children* (2008), pp. 133–136. 2

[PH08] PAULSON B., HAMMOND T.: Paleosketch: accurate primitive sketch recognition and beautification. In *IUI '08: Proceedings of the 13th international conference on Intelligent user interfaces* (2008), pp. 1–10. 5

[SD04] SEZGIN T., DAVIS R.: Scale-space based feature point detection for digital ink. In *Making Pen-Based Interaction Intelligent and Natural, AAAI Spring Symposium* (2004). 5

[SD05] SEZGIN T., DAVIS R.: Hmm-based efficient sketch recognition. In *IUI '05: Proceedings of the 10th international conference on Intelligent user interfaces* (2005), pp. 281–283. 2

[TH10] TAELE P., HAMMOND T.: Lamps: A sketch recognition-based teaching tool for mandarin phonetic symbols i. *Journal of Visual Languages & Computing 21*, 2 (2010), 109–120. 2

[TPH09] TAELE P., PESCHEL J., HAMMOND T.: A sketch interactive approach to computer-assisted biology instruction. In *IUI '09 Workshop on Sketch Recognition* (2009). 2

[YSvdP05] YANG C., SHARON D., VAN DE PANNE M.: Sketch-based modeling of parameterized objects. In *SBIM* (2005), pp. 63–72. 2