

Interpreting sloppy stick figures by graph rectification and constraint-based matching.*

James V. Mahoney
Markus P. J. Fromherz

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304

Abstract

Machine systems for understanding hand-drawn sketches and diagrams must reliably interpret curvilinear configurations that are sloppily drawn and highly variable in form. We propose a two-stage subgraph matching framework for sketch recognition that can accommodate great variability in form and yet provide efficient matching and easy extensibility to new configurations. First, a rectification stage corrects the initial data graph for the common deviations of each kind of constituent local configuration from its ideal form. Matching is then accomplished by a straightforward constraint-based subgraph matching scheme. We explore the approach in the domain of human stick figures in arbitrary poses.

1 Introduction

Machine systems for understanding hand-drawn sketches and diagrams must reliably interpret common curvilinear configurations, such as arrows, geometric shapes, and conventional signs and symbols. These figures are often sloppily drawn and highly variable from one instance to the next. We would like a recognition system to accommodate considerable sloppiness and variability in form, but without sacrificing matching efficiency or easy of extensibility to new configurations. This paper proposes a framework for meeting these design requirements and explores it in the domain of human stick figures in arbitrary poses.

Figure 1 illustrates what we mean by “sloppiness” in the stick figure context. In the neat example on the left, the lines representing the figure’s limbs are neatly drawn and meet precisely at the junctions and corners. In the sloppy example, however, the

lines, more often than not, fail to be co-terminal, due to overshoot or undershoot in the drawing process. It is easy to see even from this simple example that basic image analysis techniques alone (thinning, tracing, and detection of junctions and corners) would not provide an adequate basis for reliable recognition in this case. The stick figure domain is a good one for exploring this issue because it is just complex enough to discourage an approach in which specialized and detailed matching routines must be written for each new configuration.

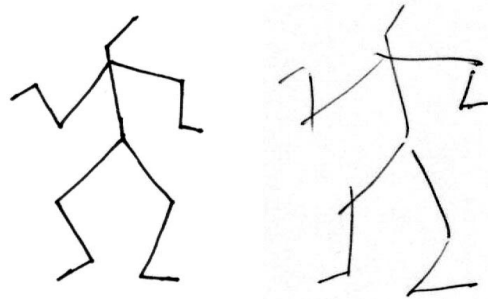


Fig. 1. Neat and sloppy stick figures. When people draw quickly, failures of co-termination are a frequent variation from the ideal form.

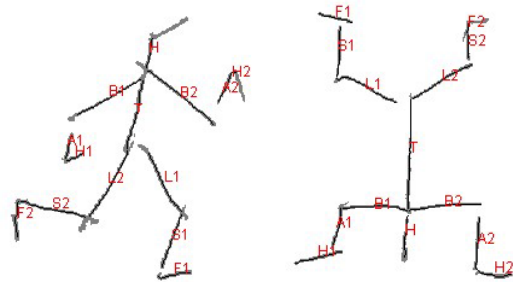


Fig. 2. Example matching results. The labels denote the model parts: Head, Torso, Biceps1, Arm1, Hand1, Thigh1, Shin1, Foot1, Biceps2, etc.

Sloppiness comes down to the fact that no one can draw the same figure precisely the same way twice, try as they might; there is always noise in the drawing process. In addition to sloppiness, there are several other pervasive sources of the variability with which a matching process must contend. There may be noise in the process of sensing what has been drawn, e.g., creating gaps in the lines or introducing extraneous marks. The class of figure

* Published at Fourth IAPR Int. Workshop on Graphics Recognition (GREC'01), Kingston, Ontario, Canada, Sept. 2001.

to be recognized may be articulated or otherwise inherently variable in shape. Finally, interaction of the figure with background context, such as overlap or occlusion (see Figs. 3, 4, 5, 6) may significantly affect the form of the description of the figure given to the matcher.

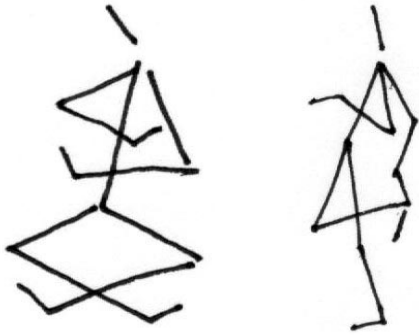


Fig. 3. Stick figures exhibiting inherent ambiguity due to self-crossings. It is locally ambiguous whether or not a figure part continues across a junction.

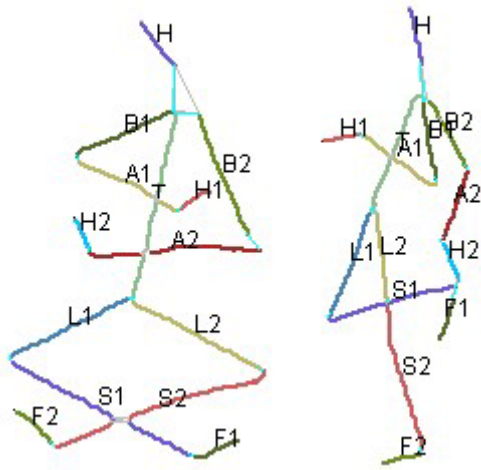


Fig. 4. Matching results for Figure 1. Simple segments are drawn in the same color if they were interpreted as a single curve.

Because the configurations found in diagrams and sketches are often highly articulated or abstract, this recognition problem is well suited to a structural modeling approach. Specifically, the configuration model and the input scene are represented as attributed graphs, with nodes representing figure parts (e.g., lines), and edges representing part relations (e.g., line connections). The attributes on

node and edges represent geometric properties associated with the underlying lines and line relations. Recognition is cast as subgraph matching of the *model graph* to the *data graph*; this allows for background context in the input scene. Attributed subgraph matching may be implemented using constraint propagation and heuristic search.

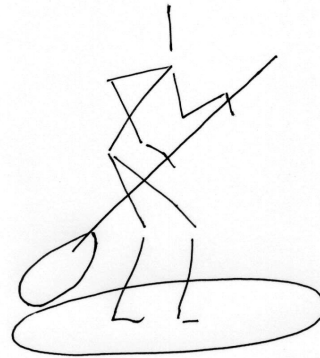


Fig. 5. A stick figure exhibiting ambiguity due to interaction with background context.

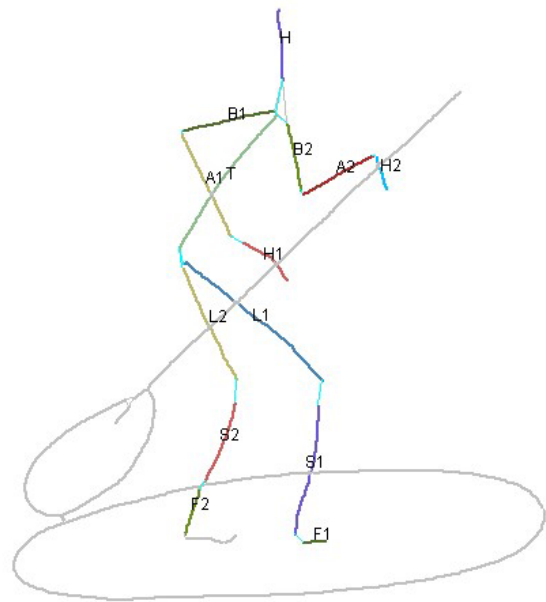


Fig. 6. Matching results for Figure 3. (Foot 2 is mislabeled due to a colinearity failure.)

A stick figure configuration *model* is expressed in a simple syntax, illustrated below. Each limb statement defines a part of the figure. The optional modifier allows a part to be missing in the data. The linked statement asserts an end-to-

end connection between two curvilinear parts. Two optional integer arguments allow the modeler to specify with which end of each part the link is associated. For example, the (default) values (2,1) indicate that the link goes from the second end of the first named part to the first end of the second, where “first” and “second” are arbitrary but must be consistent.

```

model stick_figure {
  limb head, torso, biceps1, ...;
  optional limb hand1, hand2, ...;
  link(head, torso);
  link(torso, biceps1, 1, 1);
  ...
  minimize (torso.len-2*head.len)^2
            + (2*torso.len-3*biceps1.len)^2
            + ...;
  ...
} // end model stick_figure

```

The modeling syntax allows constraints on part attributes to be specified. For example, the two minimize statements in this example specifies optimal relative limb lengths.

Consider an initial data graph, G_D , created from an image of a line drawing like Fig. 1. The nodes represent the ends of curve segments that result from applying standard binarization, thinning, junction detection, corner detection, and curve tracing operations. Two nodes are linked by an edge if their curve segments terminate at the same junction or corner. Due to drawing variability and noise, the resulting graph would rarely contain a verbatim instance of the model as a subgraph.

One solution is to use error-tolerant subgraph matching to explicitly allow and account for structural or attribute discrepancies. The matching process searches for a mapping that minimizes the so-called *edit distance* between the model and any data subgraph, which reflects predefined costs associated with particular discrepancies. However, this increases matching complexity; e.g., from $O(mn)$ to $O(mn^2)$ in the best case; from $O(m^n n^2)$ to $O(m^{n+1} n^2)$ in the worst case; m and n being the node counts of the data and model graphs respectively [Mes95]. Typically, this added cost is incurred for every model matched.

As an alternative to error tolerant matching, we propose a two-stage approach to matching an ideal

model graph to non-ideal data. The first stage, termed *graph rectification*, will explicitly correct the data graph for each of the possible deviations of a constituent local relation from its ideal form, e.g., failures of co-termination (see Figs. 7, 8). This process is an application of general *perceptual organization* principles, such as good continuation and co-termination, to the specific goal of producing a data graph in which the model is much more likely to find a direct match.

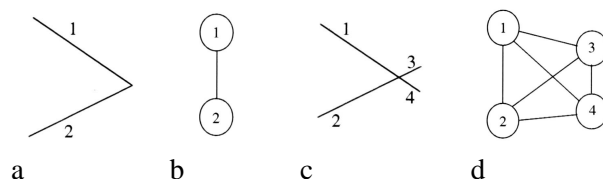


Fig. 7. Two lines just meeting at a corner (a) give data graph (b), but overshoot (c) results in graph (d). Graph normalization operations applied to (c, d) produce a graph identical to (b).

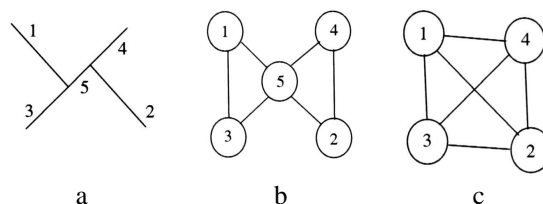


Fig. 8. Input (a) gives initial data graph (b). If segment 5 is unambiguously spurious, it is eliminated to produce graph (c). Otherwise (c) is passed on to the

We distinguish three classes of graph rectification operations: augmentation, reduction, and elaboration. *Augmentation* operations add new nodes or edges to G_D . E.g., in the *proximity linking* operation, an edge may be added where two free ends just failed to meet. Another augmentation operation, *virtual junction detection*, establishes salient proximity relations between free ends and interior curve points.

Reduction operations remove elements from G_D . E.g., the *spurious segment elimination* operation removes nodes that correspond to short curve segments arising from other segments just failing to precisely coincide at a common junction. When such a segment is removed, associated junctions are

merged, and appropriate new co-termination links are added to G_D .

Elaboration operations add subgraphs to G_D that constitute alternative descriptions of substructures already in G_D . E.g., if two segments in G_D satisfy a smooth continuation constraint, the *continuity tracing* operation may add a new node to G_D that represents the alternative interpretation of them as a single smooth curve. Graph elaboration addresses the problem of local ambiguity.

The rest of this paper is organized as follows. Section 2 details methods for constructing and rectifying the data graph. Section 3 develops a constraint-based scheme for subgraph matching. Section 4 presents experimental results. Section 5 offers concluding remarks.

2 Data graph construction and rectification

Graph construction

The initial data graph is built using standard image processing operations. First the image is binarized and thinned, and all simple curve segments are extracted by tracing. The curves are then split into smooth segments at salient corner points. (Defining perceptually significant corners computationally, and detecting them reliably, are challenging scene analysis problems in their own right that are for the most part still open. The technique we use is based on that described in [Sau95] with a free parameter P_I .) For each of the resulting segments, its graph representation is added to the initial graph.

At this point we move to a more structured view of the graph representation of a segment. It consists of *two* nodes, one for each end point, connected by a type of edge we call a *bond*. For each pair of curves terminating at the same junction or corner, another type of edge, called a (*co-termination*) *link*, connecting their co-terminal end nodes, is added to the graph. The data graph then, consists of a single type of node, representing a curve segment end point, and two types of edges (links and bonds). In this paper all graph edges are unidirectional.

This initial graph is a natural description of a curvilinear input scene for matching purposes in

that it mirrors the form of our configuration models; i.e., it makes explicit the end-to-end connectivity relations among line segments. In the face of variability due to noise, sloppy drawing, and context, however, it has three key potential shortcomings (Fig. 9). First, it may be *insufficiently connected* due to failures of one line to terminate precisely at an end point or interior point of another line by a drawing undershoot, leaving gap. Second, it may contain *spurious segments* arising from: (i) failure of one line to terminate precisely at the end of another line by a drawing overshoot, creating a spurious crossing or T-junction; (ii) failure of two lines to terminate precisely at the same point of a third line; and (iii) thinning artifacts. The third possible inadequacy of the initial graph is that it may be *over-segmented*, in that its segments may be in many-to-one rather than one-to-one correspondence with lines of the model, due to intersections of figure lines either with themselves or with background lines.

Graph rectification operations are introduced below to address each of these problem.

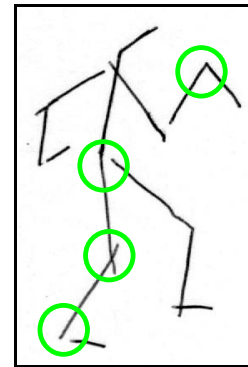


Fig. 9. This figure requires (circles, top to bottom) corner detection, virtual junction detection, junction detection and spurious segment elimination, and proximity linking.

Graph augmentation

To address the problem of insufficient connectivity, the initial graph is augmented by adding links to it. One extreme possibility would be to simply make the data graph totally connected, but this could have a negative effect on matching performance, perhaps prohibitively so. Intuitively, the ideal situation is for the data graph to contain all links required by

the model and as few additional redundant links as possible; i.e., we want the data graph to “look just like” the model, to minimize matching complexity. The obvious criterion for adding links selectively is proximity.

End-to-end proximity linking. A proximity link is an edge added to the data graph in virtue of proximity between two curve free ends. Proximity links are added to the graph in increasing order of length (i.e., distance between the ends) until the graph becomes sufficiently connected. As candidates for addition, we may use the set L of all n^2 possible links between n free ends. There are various possible optimizations on the candidate set that produce a similar result. One possibility is the edges of the Delaunay triangulation of the set of free ends. For n free ends the complexity of computing these candidates is $O(n \log(n))$. Alternatively, we may use those links in L between end points whose underlying connected components in the image are Voronoi neighbors. The results in this paper are based on the latter approach.

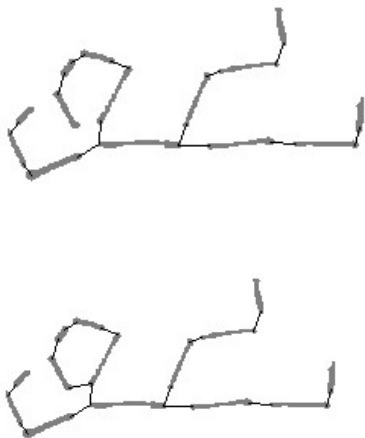


Fig. 10. Minimum spanning tree and augmented MST of a configuration.

The process of adding links is a modification of Kruskal’s algorithm for constructing the minimum spanning tree (MST) [Cor90], resulting in a ‘close-to-minimal spanning graph’ of the base graph. (The MST itself would be insufficiently connected for matching: it would exclude some structurally important proximity relations, since it cannot

contain a loop; see Fig. 10.). At each step a candidate link l is added either if (i) it is between two ends that are not yet connected (as in Kruskal’s algorithm) or (ii) at least one of l ’s ends e is not associated with a previously added link, and the length of l is within a factor P_2 (a free parameter), of the length of the curve segment containing e . (Figure 4 shows an example.) For n candidate links the complexity of this step is $O(n \log(n))$.

The spanning graph’s drawback is that it may contain long links between segments that we clearly perceive to belong to distinct proximity groupings. The matching process would ordinarily ignore such links due to a preference for shorter links, but linking the whole scene into a connected graph unnecessarily increases the cost of matching. The extension of the proximity linking process to exclude such links is termed *proximity grouping*; it is a topic of our ongoing research.



Fig. 11. Link closure “normalizes” data graphs for variations in local link configurations due to relative link lengths.

End-to-interior proximity linking (virtual junction detection). A virtual junction is a non-end point of a segment s that is sufficiently near a free end e and whose distance to e is a local minimum among points of s . The graph is modified to reflect a virtual junction j by splitting s into two segments s_1, s_2 , and adding co-termination links relating s_1, s_2 and the segment containing e . Virtual junctions may be detected using the Delaunay triangulation of the union of the free ends and evenly sampled non-end curve points. A sampled point is a virtual junction if it has a Delaunay edge to a free end e and its distance to e is (i) minimal across all other sampled points on s and (ii) within factor P_2 (above) of the length of e ’s segment. The complexity of this step is $O(n \log(n))$, for n sampled curve points. Virtual junctions may be detected, without sampling and computing the Delaunay triangulation, by searching the points of

all curves for one that minimizes distance to e and satisfies constraint (ii) above. This is the approach used in our current implementation.

Link closure. Even after proximity linking, the graph may still be insufficiently connected due to differences in relative link lengths between the input instance and the model. E.g., while the model may specify that the head and arms are each linked to the top of the torso, the proximity relations in the data may result in some quite different configuration (e.g., see Fig. 11). We address this problem by computing the transitive closure of data graph links and then augmenting the data graph with those links in the closure that it does not already contain.

Graph reduction

Spurious segment elimination. A spurious segment s results from two segments just failing to coincide at a common junction. This creates two nearby junctions in the skeleton, bridged by s . When s is removed from the graph, the junctions are merged into one, and appropriate new co-termination links are added.

The challenge, of course, is in deciding which segments are spurious. This is a classic scale selection or noise estimation problem, and experience suggests that it will be difficult to address in a definitive way. We have implemented a simple, *ad hoc* technique for the time being, while considering more rigorous methods in our ongoing work. In this *local elimination* technique, a segment is removed if (i) its length is below a free threshold parameter P_3 ; or (ii) it is shorter than any other segment terminating at a common junction and its length is within some factor (parameter P_4) of the length of the next longest segment terminating at a common junction.

Above a certain length, a spurious segment must fall outside the scope of this, and perhaps any, scaled-based, bottom-up segment elimination technique. The elimination of longer spurious segments may in some cases be effected through colinearity grouping with neighboring curves (see Fig. 12), by the method of the next subsection.

Graph elaboration

Graph elaboration deals with local ambiguity in the input. For recognition by graph matching, there is local ambiguity whenever some part of the scene could be represented in multiple ways in the data graph. We make a distinction, however, between weak and strong forms of local ambiguity, as functions of the particular matching process used.

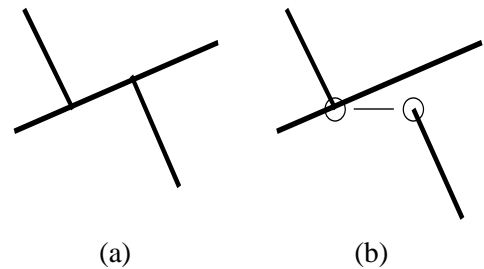


Fig. 12. (a) Longer spurious segments may not be subject to direct elimination. (b) After continuity grouping, however, proximity linking may in some cases properly associate the free line ending with the junction.

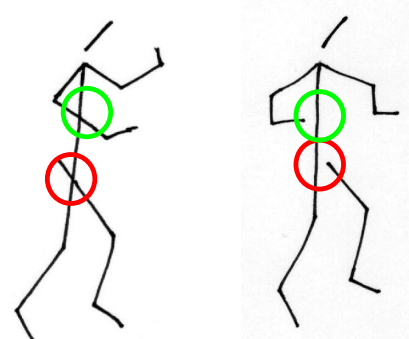


Fig. 13. Strong local ambiguity: within the lower circles the curves should be broken at a junction/virtual junction. Within the upper circles they should not.

An ambiguity is weak if passing only one of the possible descriptions to the matcher does not preclude a correct global interpretation. E.g., two segments meeting at a corner are always related by an edge in the data graph. The ambiguity here is that the edge suggests a structural association, whereas the segments might coincide only by accident. However, our matcher does not require

every data edge to match a model edge, so this is a weak ambiguity.

An ambiguity is strong if multiple descriptions must be passed to the matcher to ensure a correct interpretation; e.g., this is the case for crossing lines (e.g., see Fig. 13). The elaboration process may associate preference rankings with the alternative structures introduced, to direct the matching search toward preferred interpretations.

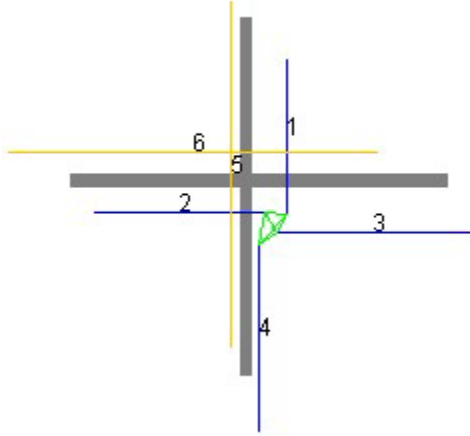


Fig. 14. The elaborated graph for two crossing lines. Blue lines: primitive segments. Yellow lines: length 2 chains. Numerals are unique segment identifiers. Green lines are links between segments.

We now introduce one particular elaboration process that addresses the ambiguity associated with smooth continuation of lines at crossings and T-junctions.

Continuity grouping. Recall the initial data graph, G_D , as described above. Suppose a local colinearity relation is established between co-terminal curve segments s_1, s_2 . (Currently we do this simply by applying an angle threshold -- free parameter P_5 -- to the angle between their co-terminal ends). A new segment, s_3 , is formed by merging s_1 and s_2 , and it is added to G_D , along with links connecting s_3 to any other nodes to which s_1, s_2 are connected. Specifically, the graph representation of s_3 consists of copies of the two non-co-terminal nodes of s_1, s_2 , connected by a new bond. The copy n_b of a node n_a is in turn assigned copies of all the links of n_a (but not of n_a 's bond). Each link of n_a and its copy in n_b are related by mutual exclusion; i.e., a valid match of a model graph to G_D may not involve more than one of these edges. This fact is made explicit by

assigning these links a common label: the unique identifier of the mutual exclusion set to which they both belong. This mutual exclusion relation applies transitively if n_b is itself copied later.

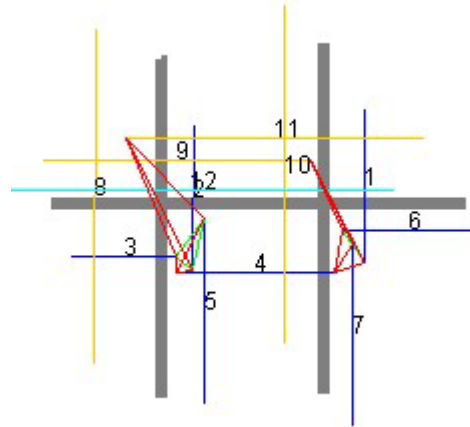


Fig. 15. The elaborated graph for three crossing lines. Cyan line: length 3 chain. Red lines are links that belong to mutual exclusive sets.

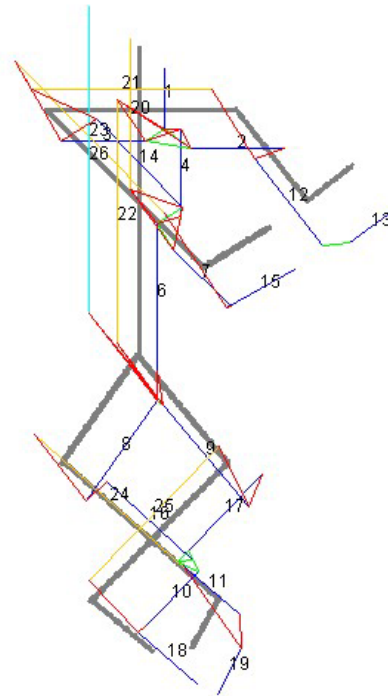


Fig. 16. The elaborated graph for a stick figure schematized for clarity.

A step of *colinearity tracing* involves applying the preceding operation to all pairs of collinear segments at every junction in the scene. By

iterating this tracing step to convergence, all sequences of pair-wise collinear segments are added to G_D . Figures 14, 15, and 16 show examples of data graphs elaborated in this way.

The idea of adding alternate subgraphs to the data graph, with a mutual exclusion relation holding among links that the alternates have to common nodes, is general. We expect to apply this approach in the future to other potentially ambiguous local relations, including co-termination at corners and colinearity at gaps.

3 Constraint-based subgraph matching

Formulation as a CSP

Our subgraph matching problem may be formulated as a constraint satisfaction problem (CSP) as follows. Mirroring the data graph, the graph representation of a limb in the model is a pair of nodes, representing the limb's ends, connected by a bond. A link statement in the model specification gives rise to (two unidirectional) links between the specified nodes.

A CSP variable is defined for each node in the model graph. The initial domain of each variable is the set of data graph nodes plus the label *null* for a missing part, since our models may specify some parts as optional.

The primary constraint of the problem, termed *link support*, is that a link/bond, l , between two model nodes, m_1 , m_2 , requires the existence of a corresponding link/bond between the associated data nodes d_1 , d_2 . This requires some spelling out because of the possibility of missing parts. Specifically, if l is a bond, then the constraint is satisfied if (i) m_1 and m_2 are both null; or (ii) d_1 and d_2 are connected by a bond. If l is link, then the constraint is satisfied if one of four conditions hold: either (i) m_1 and m_2 are both null; (ii) m_1 is null and d_2 has no links; (iii) m_2 is null and d_1 has no links; or (iv) d_1 and d_2 are connected by a link. (The reason for conditions (ii) and (iii) is that a model part should not be assigned null when there is support for it in the data.)

The *unique interpretation* constraint requires that each data node may be assigned to at most one

model node. This is implemented as a global cardinality constraint.

These two constraints are sufficient to establish a purely topological match between the model and data graphs. This is often adequate for matching to an isolated instance, but spurious matches would arise if the data contained noise, self-intersections, or background clutter. To find meaningful matches in such cases, geometric or other criteria that rank topologically equivalent solutions must also be applied. We have explored three such criteria.

The *minimal total link length* criterion prefers interpretations with smaller total length of relation links involved in the match. (A link is involved in the match if it corresponds to a link in the model.) The *optimal part proportions* criterion prefers interpretations in which the length ratios of the segments closely approximate those specified in the model. The *maximal part count* criterion prefers matches that leave fewer model parts unassigned; this criterion is needed because the constraints above allow optional model parts to be missing and make no distinction between solutions with differing numbers of missing optional parts.

Implementation

One of our implementations of the constraint solving for the above problem builds on a standard heuristic state-space search framework, that supports a variety of specific search schemes, such as A* and branch-and-bound [Rus95]. A state in the search space consists of an assignment to the set of CSP variables. (In the initial state, all variables are unassigned. In a goal state, all variables are assigned in such a way that all constraints are satisfied.) The *successor function* of such a search process is given a state taken from the search queue and returns a set of new states to be added to the queue. (These states constitute nodes of the search tree.) The successor function for a CSP problem selects an unassigned CSP variable and creates a new state for each possible assignment of this variable, applying the specified constraints in the process to effect possible reductions in the set of new states generated.

In this design, the additional criteria may be applied by incorporating them into the heuristic function (or

“objective” function) that is optimized by the search process. The three optimization criteria are combined into a single function as a weighted sum (the weights being free parameters P_6, P_7, P_8). In our current implementation, the link length and part proportion criteria have equal weights while the part count criterion has a much higher weight; i.e., the former two criteria only come into play among solutions with equal part counts.

The preceding formulation will find only a single instance of a given model in the data graph. To find multiple instances or to match multiple models, individual matches are done sequentially, with removal from the data graph of the matched nodes and any associated links/bonds at each step.

We have also implemented this basic matching scheme in a concurrent constraint programming formulation, expressed using the `clp(FD)` library and corouting in SICStus Prolog [Car97]. A detailed description of this implementation is given in [Fro01]. The two implementations give similar performance; potential optimizations may, however, have a significant effect on their relative performance. The experimental matching runtimes given in this paper are for the SICStus Prolog implementation.

Extension to elaborated graphs

The extension of this matching scheme to handle ambiguity is straightforward. Two global mutual exclusion constraints and an objective function term are added. The first constraint enforces mutual exclusion among links that are in the same mutual exclusion set. The second constraint enforces mutual exclusion among data subgraphs that have primitive segments in common. E.g., if a given segment is assigned to a model part, no other segment built from any of the same primitive segments may be assigned to any model part. The objective function term (free parameter P_9) essentially preference ranks curve segment chains according to the number of segments they contain. This respects the fact that long smooth curves are normally perceived as more salient than their constituent segments.

4 Experimental results

This section presents three empirical evaluations of the recognition technique. The first informally examines performance, on isolated figures, with respect to the specific forms of local perturbation the rectification stage is designed to address. Secondly, we investigate the effect on recognition runtime and accuracy of adding distractor lines at random to an isolated figure. Finally, we explore runtime and accuracy on inputs containing multiple instances of a single model.

Recognition accuracy on isolated figures

We explored the effects of the various types of local variation, both independently and in combination, on stick figures in a wide variety of postures. We developed five parallel example sets, A through E, of 20 figures and 4 non-figures each (Table 1). These sets are shown in Figs 23 through 27, with the non-figure examples excluded. An example of the non-figures for one of them is shown in Fig. 17.

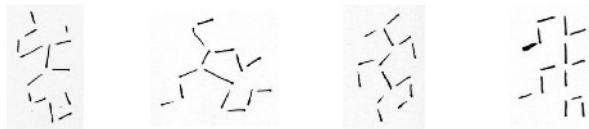


Fig. 17. Non-figure examples from the test set of Fig. 24.

This is an informal evaluation, as many of these same figures were also used during the development process; they were all drawn by one of the authors; and the scoring is somewhat subjective. Nevertheless, they illustrate the ability of the matching scheme to cope with highly variable configurations of a given model, and with varying degrees of slop. The matcher was run with the objective function turned off. The results and scoring system are shown in Tables 2 and 3.

This assessment also provides a basis for interpreting the results of the subsequent experiments, which define “error” in terms of agreement with a base match. The test cases for those experiments were generated using the images of example set B only. We see from Table 2 that

the base match for set B is indeed correct in all but one case. (Another reason for basing further experiments on set B was that all data graph links are readily displayed in the case of disconnected figures.)

Set	Type of variation	Examples
A	Connected, neatly drawn	Fig. 23
B	Disconnected limbs	Fig. 24
C	Overshoots	Fig. 25
D	All variations (neater)	Fig. 26
E	All variations (sloppier)	Fig. 27

Table 1. Test sets used.

Set	Correct	False+	False-	Fail	Partial
A	22		2		
B	23			1	
C	21		3		
D	17	2	2	1	2
E	20		2		2

Table 2. Recognition performance on isolated or nearly isolated examples.

Score	Interpretation
Correct	every limb labeled correctly
False+	a non-figure labeled as a figure
False-	no solutions for a valid figure
Failure	most limbs mislabeled
Partial	head, torso, most limbs labeled correctly

Table 3. Result scoring system.

The effect of the objective function

The preceding results show that matching algorithm can usually correctly interpret isolated figures without the objective function. However, in the presence of noise or background clutter, as in Fig. 18a, optimization is essential. For example, Fig. 19 displays the model instances found in Fig. 18a when no or only some of the objectives are turned on. In the following experiments, all objective function terms are turned on.

The effect of random distractor lines

We conducted a series of experiments to measure how the algorithm scales with an increasing number of distractor lines. For a set of twenty stick figure drawings (as shown in the various figures of this

paper), we added from 0 to 29 random, non-overlapping lines, each combination of drawing and random-line count repeated ten times. (Note that with 29 distractor lines, the number of nodes in the data graph is triple that of the model graph.) This resulted in a total of 300 runs per drawing, or 200 runs per random-line count, for a total of 6000 runs.

Fig. 20a shows the runtimes of these experiments for an increasing number of random lines, averaged over the 20 drawings and 10 runs per drawing. (All times are given in milliseconds.) Fig. 20b shows the corresponding average error rate. The unit of error is the number of line interpretations that mismatch with the base case (0 distractors).

For no or few distractors, runtimes are typically around 0.5 to 1 s. While an average of 20 s (for 27+ distractors) is long, the overall curve shows very slow growth in runtime, attesting to the effect of constraint propagation on search. Furthermore, realistic sketches contain no more than five to ten nearby distractors, for which the increases in runtime are barely noticeable. Note however the large standard deviation. Distractors sometimes lead to almost correct stick figures in the data, literally distracting the search algorithm from the real stick figure.

Similarly, it is not surprising that the number of errors increases with the number of distractors. Sometimes, a distractor line makes a “better” limb than one in the original drawing. Still, according to the data in Fig. 20b, the error rate appears to increase only linearly with the number of distractors for these experiments.

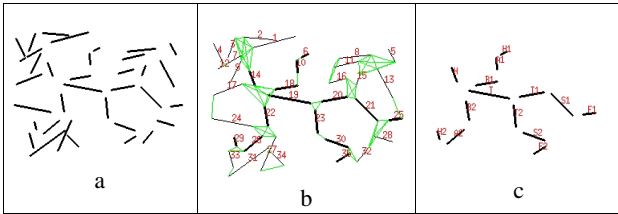


Fig. 18. A stick figure with 20 distractor lines (a), the corresponding graph with labels and links produced by the image analysis stage (b), and the interpretation found by the matching process (c).

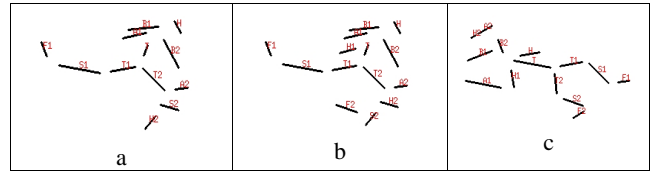
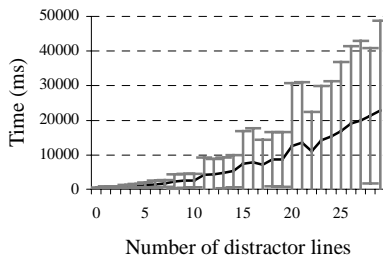
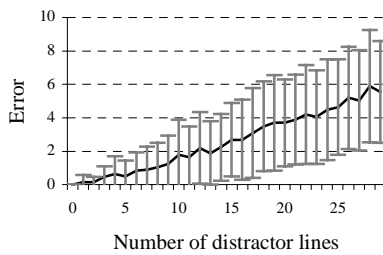


Fig. 19. Three stick figure instances found in the data of Fig. 7a with different objective functions: a) no optimization, b) maximal part count term only, c) maximal part count and optimal part proportion terms only.

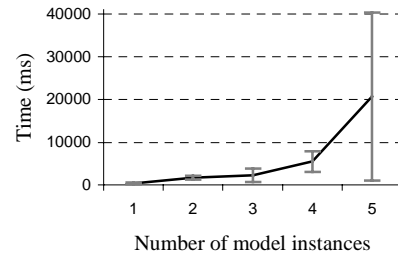


(a)

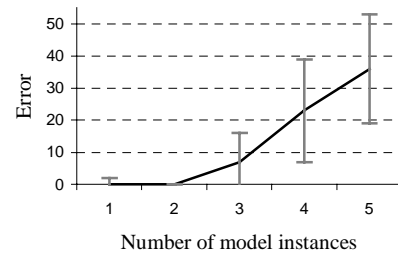


(b)

Fig. 20. a) Average runtimes to identify the stick figure in sketches with increasing numbers of random lines (0 through 29). b) Average errors in identifying the stick figure in the same runs. For each random-line-count, the standard deviation over 200 runs is shown as an error bar.



(a)



(b)

Fig. 21. a) Average runtimes to find stick figures in sketches with increasing numbers of model instances (1 through 5). b) Average errors in finding the stick figure in the same runs. For each random-line-count, the standard deviation over 10 runs is shown as an error bar.

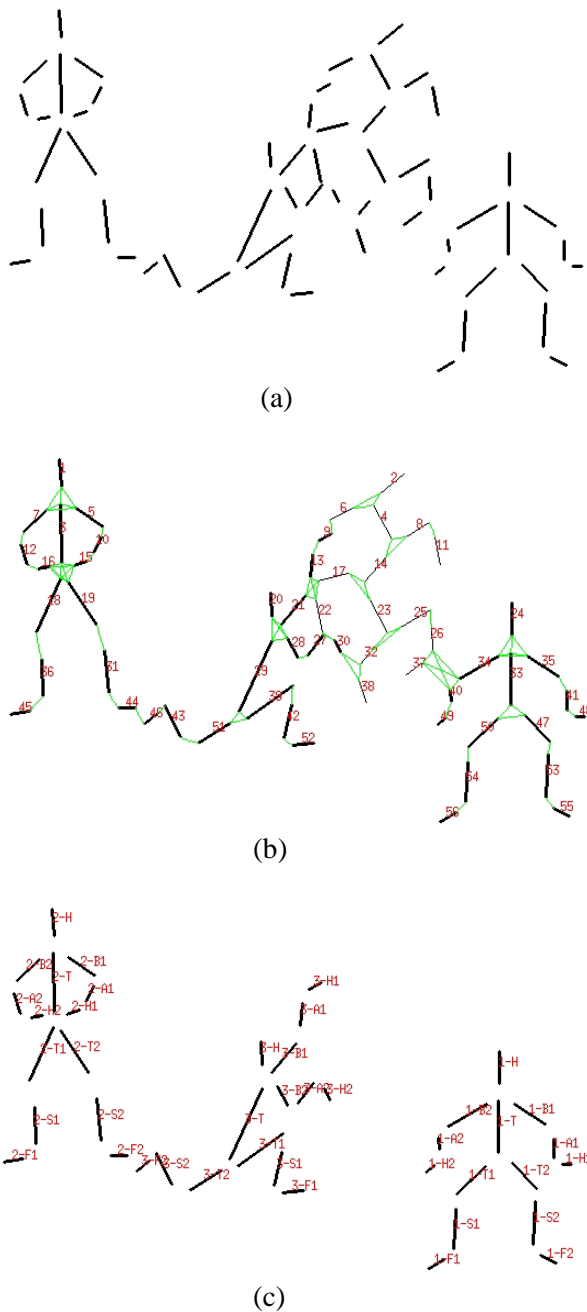


Fig. 22. Sketch with three stick figures and a distractor figure (a), the corresponding graph with labels and links produced by the image analysis stage (b), and the interpretation found by the matching process (c), with interpretation labels preceded by the index of the instance.

Performance on multiple instances

A second series of experiments measured how the algorithm scales with multiple model instances (and occasional distractor configurations). Fig. 22 shows a sample sketch with three stick figures, its data graph, and the identified model instances.

Fig. 21a shows the runtimes of these experiments for an increasing number of model instances, averaged over 10 cases. Fig. 21b shows the corresponding average error rate. The unit of error is the number of line interpretations that mismatch with the base case (component image by itself).

The runtimes appear to show exponential runtime growth in the data size. However, the average runtime for 5 instances (about 70 data nodes) is still only about as much as the average runtime for one instance plus 29 distractors in the previous experiment (about 43 data nodes). This is probably due to the fact that the model instances tend to have few interconnections with each other. Also, the error curve seems to follow a similar trend as in the first experiment. Overall, these results show that our approach should give reasonable performance for data graphs of moderate size. For large data graphs, however, there clearly is a need for additional steps to focus the matching process on appropriate subsets of a scene.

Future evaluations

Several further evaluations of this scheme are planned. First, we will measure the effect of local variation on isolated figure matching accuracy in a more systematic way, e.g., by applying random jitter to individual skeleton segments and then comparing the match of the resulting figure to the base case. Second, the effectiveness of matching under local ambiguity has not been thoroughly studied; we will explore this by adding random distractor lines that are allowed varying numbers of intersects with the figure. Third, we plan to evaluate matching of multiple models to scenes containing multiple instances.

5 Discussion

What we set out to build at the beginning of this research is a recognition system for simple, general curvilinear configurations of the kind that people routinely produce and consume in everyday graphic communication with one another. It is immediately clear that sloppiness, so characteristic of human drawing performance, constitutes a wide gulf between how people express themselves graphically and what current machine interpretation techniques can make sense of.

The technical question is: where should the knowledge and computational effort needed to counter the effects of slop reside? The answer should not be the model, for that would make extending the system to new configurations prohibitively tedious. It should also not be the matcher, we feel, because that increases matching cost for a problem that is already dangerously complex computationally, and because it limits our freedom of choice in matching machinery.

Instead, we located the knowledge about common local geometric variations and the processing to account for them in a dedicated prior process – graph rectification -- within a structural matching formulation. A constraint-based approach to matching suggests itself because it mates well with declarative modeling, and because constraint propagation and heuristic search make effective use of the natural structure of this domain. But it is important to note that matching could be accomplished using various other technologies as well; that is one of the areas we are beginning to explore in ongoing work.

Graph matching is a well-established approach to recognition of structural models and there is previous work in which explicit correction of an initial graph has been employed (e.g., [Ise86]) prior to matching. We are not aware of a previous application of this technique to the domain of hand-drawn sketches or diagrams. Messmer [Mes95] gives an analysis of possible “distortions” in input drawings similar to our analysis of local variability, but applies it in the context of error-tolerant subgraph matching. The work described in [Fah98] also uses alternate subgraphs to represent

alternative interpretations in a discrete relaxation framework.

Regarding constraint-based pattern recognition approaches, they have previously been used primarily in domains with strong (visual) grammars, such as musical notation ([Cou95],[Bai96]) and state-transition diagrams [Cho95]. The former two references extend Definite Clause Grammars to allow for the nonlinear composition of the graphical elements; the latter uses Constraint Multiset Grammars for similar reasons. Other work has proposed dedicated forward checking and full lookahead search algorithms for subgraph matching [Sha81][Lar01]. In these cases, special-purpose algorithms were developed that cannot be extended easily to user-provided constraints and objective functions.

References

- [Bai96] D. Bainbridge and T. Bell. An extensible optical music recognition system. In *Proc. Nineteenth Australasian Computer Science Conf.*, 1996.
- [Bev97] J. R. Beveridge and E. M. Riseman. How easy is matching 2D line models using local search? In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 6, June 1997, pp. 564-579.
- [Car97] M. Carlsson, G. Ottosson, B. Carlson. An open-ended finite domain constraint solver. *Proc. Programming Languages: Implementations, Logics, and Programs*, 1997.
- [Cho95] S. Chok, K. Marriot. Parsing visual languages. *Proc. 18th Australasian Computer Science Conf.*, 27(1), 1995: 90--98.
- [Cor90] T. Cormen, C. Leiserson, R. Rivest. *Introduction to Algorithms*. Cambridge, MA: M.I.T. Press, 1990.
- [Cou95] B. Couasnon, P. Brisset, I. Stephan. Using logic programming languages for optical music recognition. *Proc. 3rd Int. Conf. on Practical Application of Prolog*, Paris, 1995.
- [Epp99] D. Eppstein. Subgraph isomorphism in planar graphs and related problems. *Journal of Graph Algorithms and Applications*. 3 (3) 1999: 1–27
- [Fah98] H. Fahmy and D. Blostein. A graph-rewriting paradigm for discrete relaxation: application to sheet music recognition. *Int. Journal of Pattern Recognition and Artificial Intelligence*, Vol. 12, No. 6, Sept. 1998, pp. 763-799.
- [Fro01] M. Fromherz and J. Mahoney. Interpreting sloppy stick figures with constraint-based subgraph matching. Submitted to *7th Int. Conf. on Principles and Practice of Constraint Programming*. Paphos, Cyprus: 2001.

- [Hen91] P. van Hentenryck and Y. Deville. The cardinality operator: A new logical connective and its application to constraint logic programming. In *Proc. Int. Conf. on Logic Programming*, 1991, pp. 745-759.
- [Ise86] D. Isenor and S. Zaky. Fingerprint identification using graph matching. *Pattern Recognition*, vol. 19, no. 2, 1986, pp. 113-122.
- [Lar01] J. Larrosa and G. Valiente, Constraint satisfaction algorithms for graph pattern matching. Under consideration for publication in *J. Math. Struct. in Computer Science*, 2001.
- [Mah01] J. Mahoney and M. Fromherz. Interpreting sloppy stick figures by graph rectification and constraint-based matching. To appear: *4th IAPR Int. Workshop on Graphics Recognition*: Kingston, Ontario: Sept., 2001
- [Mes95] B. Messmer. Efficient graph matching algorithms for preprocessed model graphs. PhD thesis. Bern Univ., Switzerland, 1995.
- [Rus95] S. Russell and P. Norvig. *Artificial Intelligence: a modern approach*. Prentice Hall, 1995.
- [Sau95] E. Saund. Perceptual organization in an interactive sketch editor. *5th Int. Conf. on Computer Vision*. Cambridge, MA: 1995: 597--604.
- [Sha81] L. G. Shapiro and R. M. Haralick. Structural descriptions and inexact matching. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-3, no. 5, Sept. 1981, pp. 504-519.

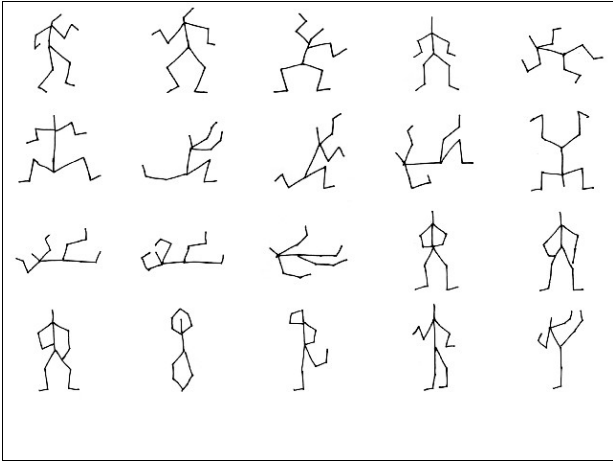


Fig. 23. Example Set A: neatly drawn.

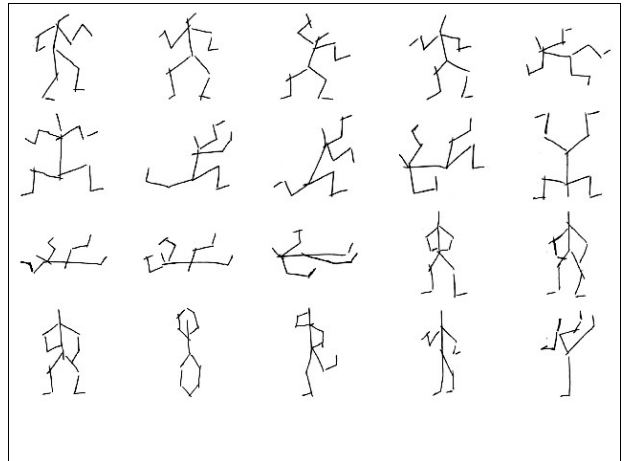


Fig. 26. Example Set D: somewhat sloppy.

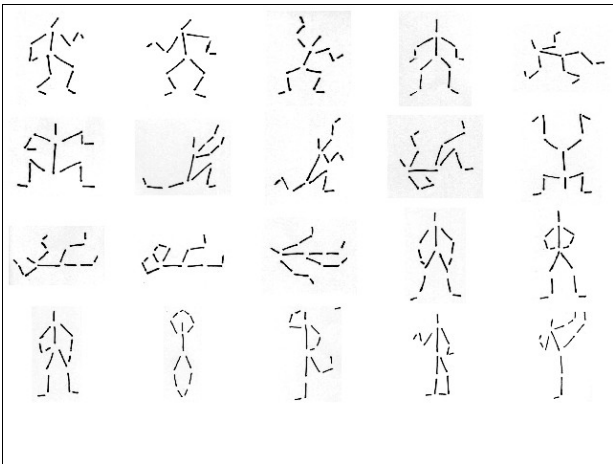


Fig. 24. Example Set B: disconnected.

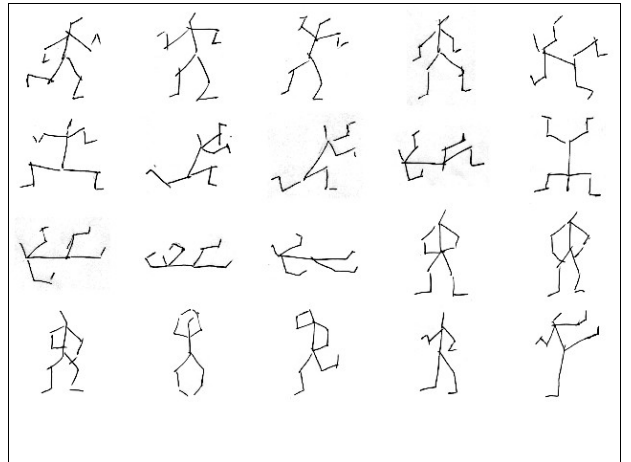


Fig. 27. Example Set E: sloppy.

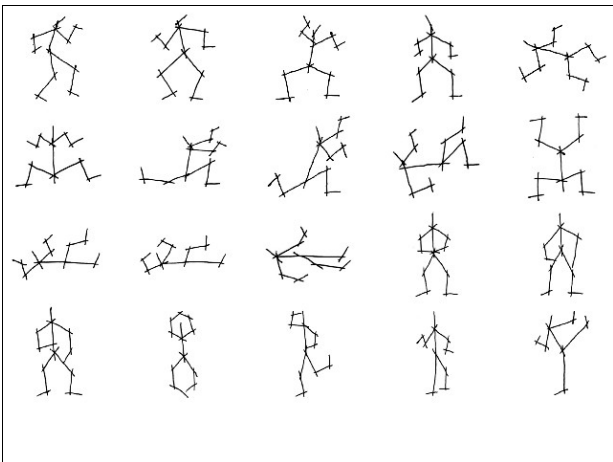


Fig. 25. Example Set C: overshoots.

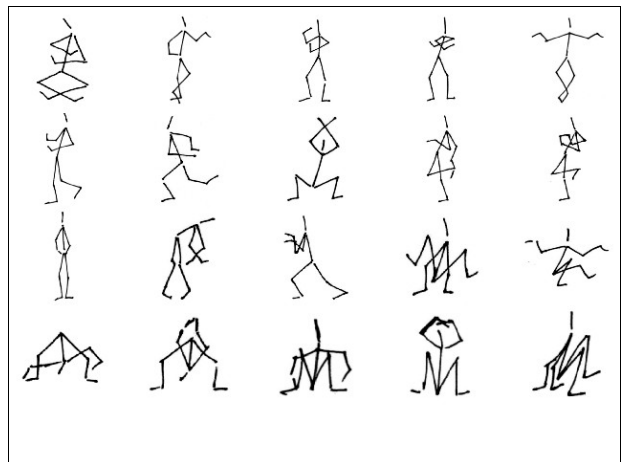


Fig. 28. Examples with self-crossings.