Automated interpretation and editing of fuzzy line drawings

by SHI-KUO CHANG†
IBM Thomas J. Watson Research Center
Yorktown Heights, New York

INTRODUCTION

One of the problems in automated line drawing analysis is to construct a cleaner drawing from the original drawing. For example, a chemist using an interactive chemical structure analysis system will usually draw a rough sketch of a chemical structure on a display device or a tablet. It is desirable to obtain a cleaner version of the rough sketch so that the cleaner drawing can be redisplayed. Moreover, the cleaner drawing can then be used to generate hard-copy output through a plotter, a magnetic film recorder or a photocomposer. Such an automated line drawing editing program can enhance the usefulness of an interactive system and is also of value in itself.

For a particular application, one usually can design a specific program to clean up a particular class of drawings. We can easily envisage a program for editing chemical structures, a program for editing a special kind of engineering drawings and so on. However, such special-purpose programs will not be able to edit a wide variety of different kinds of line drawings. In order to do so, we must design a general-purpose line drawing editing program.

In this paper we address ourselves to the problem of designing a general purpose line drawing editing program. First, we discuss the general philosophy of a table-driven line drawing editing program. The basic notion is to regard a line drawing as a fuzzy program and then to attempt to interpret the fuzzy program. The interpretation process is controlled by several tables. Therefore, the line drawing editing program is table-driven in the sense that by changing the tables the program can interpret different kinds of line drawings. It is analogous to a table-driven compiler. The main difference is that the program processed by a table-driven compiler has a unique interpretation, whereas a fuzzy program may have several interpretations and the line drawing editing program must be able to search for the correct interpretation. Next, a program for the automated editing of interconnected polygons is described. It is also shown how the weight table can be updated so that the average time of searching for the correct interpretations for a given collection of fuzzy programs is minimized. Several illustrative examples are given. Finally, some possible extensions of the present program are discussed. It is suggested that a hierarchically organized line drawing editing program may be able to interpret more complicated line drawings.

THE INTERPRETATION OF FUZZY LINE DRAWINGS

A line drawing is a collection of elementary line segments. The elementary line segments may consist of straight lines or simple analytic curves or more complicated curves. The set of all the different elementary line segments is called the vocabulary of the line drawings. For example, we may use straight lines in the $\pm 0^\circ$, $\pm 45^\circ$, $\pm 90^\circ$, and $\pm 135^\circ$ directions to construct line drawings called chain-coded line drawings. Such chain-coded line drawings have a finite vocabulary $V = \{ C_0, C_1, C_2, C_3, C_4, C_5, C_6, C_7 \}$, where $C_i$ denotes a directed line segment on a finite grid whose length is $(\sqrt{2})^q$ and whose angle referenced to the x-axis is $i x(45^\circ), q = i \mod 2$, as shown in Figure 1. Any chain-coded line drawing can be constructed using chains of elements in $V$. For example, the triangle shown in Figure 2 can be chain-coded as $C_0 C_2 C_3 C_4$.

Since for automated processing of line drawings all line segments have parameters with limited precision, the vocabulary of the line drawings can usually be assumed to be finite. The vocabulary can be thought of as the ‘building blocks’ for the line drawings. Using the line segments in the vocabulary one can construct...
Figure 1—The eight basic line segments for the chain code

infinitely many different line drawings. However, for a given application only some of those possible drawings are of interest. For example, one may wish to generate only those drawings that resemble chemical structure diagrams. In such cases we impose additional constraints on the generation process. Generally speaking, there are two kinds of constraints: (1) Prototype Generation Constraint which specifies what prototype drawings are to be created, and (2) Interconnection or Spatial Relationship Constraint which specifies what kind of interconnections can be made between prototypes or what kind of spatial relationships must be satisfied between prototypes.

For example, for the generation of chemical structures, the prototypes are polygons. They are then interconnected to form more complicated structures. Thus the Prototype Generation Constraints (abbreviated to PGC) allow only the construction of polygons. The Interconnection Constraints (IC) allow all kinds of interconnection between polygons and the construction of tree-like structures.

As another example, for the generation of mathematical expressions, the prototypes are alphanumeric characters and special function symbols. They must satisfy certain spatial relationships to form valid expressions. Thus the PGC's allow the construction of alphanumeric characters and special symbols, and the Spatial Relationship Constraints (SRC) allow the creation of valid mathematical expressions.

For the automated analysis and generation of line drawings, such constraints (PGC, IC and SRC) can best be expressed by formal descriptive grammar rules. In Reference 2 we have discussed the use of hierarchical two-dimensional grammars for the analysis and generation of drawings. In Reference 3 the problem of analyzing two-dimensional mathematical expressions is treated. It suffices to say here that formal descriptive techniques have been used with some success to analyze and generate (artificial) line drawings. However, in all cases considered, the drawings are assumed to be ideal or close to ideal. It is to the analysis of less ideal, or fuzzy, drawings that we should now focus our attention.

First of all, it is obvious that line drawings such as hand-printed or hand-written characters, chemical structures, engineering sketches are all non-exact or fuzzy. Fuzziness comes in at two levels: (1) instead of a finite vocabulary, an infinite vocabulary is used, (2) instead of well formed exact structures, non-exact structures are created. We will discuss these two points in some detail.

In the automated analysis of line drawings we assume that the vocabulary is finite. For example, in the chain-coded drawing we assume that all the line segments are in the eight prescribed directions. However, in the original drawing there might be curved line segments or straight lines in other directions. In order to obtain the chain-coded version, the original drawing has to be interpreted or, in usual terminology, quantized. Quantization is thus the process in which the original infinite vocabulary is reduced to a finite vocabulary so that subsequent analysis of the drawing can be facilitated. Quantization is usually regarded as a preprocessing operation. Nevertheless, an incorrect quantization will probably result in an incorrect analysis. Thus we prefer to call the vocabulary reduction process the interpretation process, in which the 'meaning' of individual line segments is decided by conditional (i.e., context dependent) quantization of the original line segments.

This point of view can be reformulated as follows. We can regard the original line drawing as a fuzzy program, the fuzzy instructions being the individual fuzzy line segments. Our objective is to interpret the fuzzy program so that we can construct the corresponding non-fuzzy program (non-fuzzy line drawing). Thus the problem becomes one of interpreting fuzzy programs. If the interpretation is correct, then the non-fuzzy program is executable in the sense that it does not violate any of the prototype generation constraints. If the interpretation is incorrect, then some of the con-

Figure 2—A chain-coded triangle
Automated Interpretation and Editing of Fuzzy Line Drawings

Figure 3—Automated program interpretation system

Figure 4—Equivalent theoretical model

Figure 5—Automated line drawing editing program

... strains are not satisfied and the non-fuzzy program is not executable. The second kind of fuzziness can also be taken care of using the formulation just described. The interconnection constraints and spatial relationship constraints determine the well-formedness of the non-fuzzy line drawings. Therefore, if the interpretation is correct, then the resulting non-fuzzy program (line drawing) should have a well-formed structure. An incorrect interpretation will generate a program (line drawing) having no well-formed structures.

Therefore, our objective can be stated as follows: find the non-fuzzy program corresponding to a given fuzzy program such that the resulting non-fuzzy program is executable (i.e., it satisfies all the constraints).

In summary, the system we propose is diagrammatically represented in Figure 3. The input fuzzy program is first interpreted. If the resulting non-fuzzy program is well-formed, then it is executed (i.e., the clean line drawing is sent to the output device). Otherwise another interpretation pass is entered. When all possible interpretations have been tried without success, the failure exit can be taken. This is the “No Interpretation” case.

It should be mentioned that the theory of fuzzy programs had been studied in detail. The equivalent theoretical model for the system shown in Figure 3 is illustrated in Figure 4. The (non-deterministic) finite-state machine represents the interpretive process enclosed in the dotted-line-block in Figure 3. In this particular case the programs are line drawings, and the constraints are incorporated into the finite state machine so that the machine stops in a final state if and only if the fuzzy program satisfies all the imposed constraints. In Reference 4 we have treated the problem of executing fuzzy programs using finite-state machines.
We have shown that if the fuzzy program is regular, then one can decide whether it has a non-fuzzy execution and, in case it does, one can effectively construct the corresponding non-fuzzy program. Therefore, from a theoretical point of view, we can always interpret (or reject) a fuzzy program provided that it is regular. Since line drawings can always be regarded as regular expressions, theoretically their interpretations can always be found. However, from a practical point of view, we not only want to find the correct interpretation but also would like to find it in the shortest time possible. In other words, efficiency in finding the correct interpretation is also of importance. This problem will be discussed in the next section.

AN EXPERIMENTAL PROGRAM FOR LINE DRAWING INTERPRETATION

In the above we have discussed the general concepts of fuzzy program (fuzzy line drawing) interpretation. In this section an experimental program for the interpretation of interconnected polygons will be described.

The experimental line drawing interpretation program is organized as shown in Figure 5. The program is initialized by reading in a table of (preconceived) prototypes and another table of initial weights for fuzzy instruction interpretation. Once properly set up, the program can accept fuzzy line drawings and attempt to interpret them. If the interpretation is successful, then the weight table will be updated so that successful interpretations are reinforced. The overall effect of the updating procedure is such that the average search time for a given collection of fuzzy programs will be minimized. Thus we may say that the program is able to learn the idiosyncracies of the individual user and tries to adjust its interpretation of his drawings accordingly.

In the following paragraphs we describe the experimental program in some detail.

First we describe the four basic tables of the program.

1. INPUT TABLE: An array of size N by 5, where N is the number of line segments of the input drawing. In any row, the first entry contains a pointer to the corresponding line segment in the output table (initially 0), the remaining four entries are the x and y coordinates of the starting and terminating points of the line segment. See Figure 6(a).

2. OUTPUT TABLE: An array of size M by 8, where M is the number of line segments of the output drawing (≠N in general). In any row, the entries are SWH (1 if fixed and 0 if free), NMBR (k if this line segment is in the kth polygon), NAME (a pointer to the corresponding line segment in the input table), X1, Y1, X2, Y2 (x and y coordinates of the starting and terminating points of the line segment) and ANGLE.

3. PROTOTYPE TABLE

4. CONDITION TABLE

Figure 7

From the collection of the Computer History Museum (www.computerhistory.org)
(1) If $0^\circ$, $2$ if $30^\circ$, $3$ if $45^\circ$, $4$ if $60^\circ$, $5$ if $90^\circ$). See Figure 6(b).

3) PROTOTYPE TABLE: This table is organized as shown in Figure 7(a). Given the number of sides of a polygon, one first finds out the number of prototypes for this polygon, then one can find out the number of conditions for a particular prototype, finally the conditions for this prototype are found in the CONDITION TABLE. To see how the prototypes are described, in Figure 7(b) a number of prototypes and their corresponding descriptions are given. Each pair $(I, J)$ says in effect that side $I$ should be symmetric to side $J$ with respect to a hypothetical axis (the program will use this description to generate both x-axis and y-axis symmetric polygons), and pair $(I, I)$ simply specifies that side $I$ is perpendicular to the axis in question. We can use this description to describe axial symmetric or complete symmetric polygons.

4) INSTRUCTION SELECTION WEIGHT TABLE: This table is organized as shown in Figure 6(c). Given the state (i.e., the context in which it is located, e.g., a triangle, a rectangle, etc.) and the angle of a (fuzzy) line segment, one can search the STATE array and the ANGLE array to find out the BASE address and the DISPLACEMENT. In our case the state of a line segment is the number of branches of the polygon of which it is a part (the state is 1 if this line segment is not in any polygon). BASE+DISPLACEMENT gives us the ADDRESS of a row in the WTABLE array. The weights in this row vector determine which of the five angles ($0^\circ$, $30^\circ$, $45^\circ$, $60^\circ$, $90^\circ$) is most preferable when the fuzzy line segment has the given angle and is in the given state. The angles can be ordered according to their weights, larger weights being more preferable. One can thus construct a ranked list of angles for each fuzzy line segments, which is stored in the LK array. The last entry of the LK array is the ADDRESS of this weight vector.

Now we can explain the interpretation algorithm. The flowchart of the interpretation algorithm is shown in Figure 8. The main strategy is to “treat the polygons first, then handle the interconnections.” The program first locates a polygon in the INPUT TABLE. It then uses the INSTRUCTION SELECTION WEIGHT TABLE to construct LK tables (ranked lists of angles) for all line segments in this polygon. Thus for each LK table there is a line segment. The program next looks at all possible interpretations of the polygon by testing whether the conditions of the prototypes in the PROTOTYPE TABLE are satisfied or not. From those possible interpretations, the correct one (the one satisfying all the constraints) is selected. The angles of the line segments are now fixed. The clean polygon is next constructed with respect to the selected interpretation. It is then stored in the OUTPUT TABLE.

For example, the triangle shown in Figure 9(a) is fuzzy. Its line segments have LK tables as shown in Figure 9(b). The best interpretation is found to be $(2, 2)$, $(1, 3)$), and the clean triangle is shown in Figure 9(c). Notice that the angles are first determined so that the orientation of the polygon becomes fixed. The location and length of the line segments can then be deter-
mined by rotating, extending or shrinking the original line segments.

After a polygon has been successfully interpreted, WTABLE is updated so that such an interpretation is reinforced. The updating procedure is the following. Suppose \( l_1, l_2, \ldots, l_n \) are the line segments that have just been interpreted. Let ADDRESS\(_i\) be the entry in WTABLE corresponding to \( l_i \), ANGLE\(_i\) be the correct interpretation of the angle of \( l_i \), and COST\(_i\) be the search cost incurred if the angle of \( l_i \) is not interpreted as ANGLE\(_i\). Then for all \( i, 1 \leq i \leq n \),

\[
\text{WTABLE}(\text{ADDRESS}_i, \text{ANGLE}_i) \leftarrow \text{WTABLE}(\text{ADDRESS}_i, \text{ANGLE}_i) + \text{COST}_i.
\]

In other words, if the interpretation of the angle of \( l_i \) is ANGLE\(_i\), then \( \text{WTABLE}(\text{ADDRESS}_i, \text{ANGLE}_i) \) is increased by \( \text{COST}_i \). Thus correct interpretations are reinforced. The updating procedure is simple and intuitively appealing. Its theoretical justification can be found in Reference 4, where it is shown that this procedure actually minimizes the average search time for a given collection of fuzzy programs.

The program repeats the above procedure for every polygon in the line drawing. Since the polygons may touch one another, some sides of the second (or third, \ldots) polygon may have already been determined. In this case they are treated as fixed (i.e., SWH = 1 in its row in OUTPUT TABLE) and a similar procedure is carried out for the free line segments.

When all the polygons have been treated, the program then handles the interconnection line segments. Notice that since the 'state' is now different, by changing the BASE address one can use a different WTABLE for those line segments. For example, only 0°, 45°, 90° lines are allowed. Thus although the angles are the same, with different contexts (different states) the interpretation can be different. When the interconnection line segments are fixed, the position and size of the polygons can be adjusted, so that no conflict occurs (no overlapping polygons, no lines piercing into polygons, etc.) in the clean line drawing.

The present version of the experimental program, written in FORTRAN, is able to interpret interconnected polygons. Some of the results are illustrated in Figure 10 and Figure 11. The drawings are produced by a plotter. The drawings shown on the left are the originals and those on the right are the clean versions. It is clear that the polygons are symmetric with respect either to the x-axis or to the y-axis, and the interconnected lines have also been adjusted. One obvious application of this program is the editing of hand-drawn chemical structures. However, since the prototype table can be modified at will, the program is not restricted to
this application and may be used to interpret other kinds of line drawings. For chemical structure editing, in the last phase one can insert alphabetical characters into the line drawing to make it complete. This phase is also indicated in the flowchart (see Figure 8), but has not been incorporated into the experimental program.

DISCUSSIONS AND CONCLUSIONS

In the previous sections we have described a general approach to fuzzy line drawing interpretation and a specific experimental program for the interpretation of interconnected polygons. The basic notion of our approach is to regard a fuzzy line drawing as a fuzzy program and to attempt to interpret the fuzzy program. The interpretation is found by testing whether the fuzzy program (or part of the fuzzy program when a local interpretation is possible) satisfies certain constraints. The constraints are either Prototype Generation Constraints or Spatial Relationship Constraints. In the experimental program the first kind of constraints are incorporated into the PROTOTYPE TABLE. The second kind of constraints are built into the program: spatial conflict is avoided by adjusting the size and location of the polygons. For a more general system, it would be more desirable to state all constraints explicitly. The spatial constraints can then be coded as grammar rules and read into the program, as shown by the dotted box in Figure 5. Moreover, the flowchart shown in Figure 8 indicates that the program consists of two steps:

(1) The interpretation of prototypes.
(2) The organization of prototypes into higher level structures.

In our case the prototypes are polygons, and the higher level structures are interconnected polygons. For more complicated line drawings, one may iterate the two steps to obtain higher and higher level structures. In each level, there may be a different set of Prototype Generation and Spatial Constraints. Therefore, we have a collection of constraints organized into levels or hierarchies. This is the concept of a hierarchical grammar. In the present experimental program we have only two levels. The techniques developed are also applicable to a multi-level analysis and interpretation program. Such extension seems to be both theoretically interesting and practically useful.

In conclusion, the table-driven line drawing editing program described in this paper is but one small step toward the complete automation of fuzzy line drawing analysis. Two main concepts have been introduced. First, tables are used to control the interpretation process, so that the program can be used to interpret different kinds of line drawings. Second, a weight updating procedure is included in the program so that the average search time for a given collection of fuzzy programs can be minimized. However, the program is still not general enough to interpret a wide variety of fuzzy line drawings. It is suggested that a hierarchically organized program will be useful to interpret more complicated drawings. Experimental data should also be gathered to see whether the weight updating procedure indeed improves significantly the performance of the program. We hope that this preliminary investigation will lead to more interesting development in the future.

REFERENCES

1 FREEMAN GLASS
On the quantization of line-drawing data
IEEE Trans on Systems Science and Cybernetics Vol SSC-5 No 1 January 1969

2 S K CHANG
The analysis of two dimensional patterns using picture processing grammars
Conference Record Second ACM Symposium on Theory of Computing May 4-6 1970 Mass pp 206-216

3 S K CHANG
A method for the structural analysis of two dimensional mathematical expressions
Information Sciences 2 July 1970 pp 253-272

4 S K CHANG
Fuzzy programs—Theory and applications
Proc of the Polytechnic Institute of Brooklyn XXI International Symposium on Computers and Automata Brooklyn New York April 1971