Understanding Diagrams in Technical Documents

Robert P. Futrelle, Ioannis A. Kakadiaris, Jeff Alexander, Catherine M. Carriero, and Nikos Nikolakis
Biological Knowledge Laboratory, College of Computer Science, Northeastern University, Boston, MA 02115

Joseph M. Futrelle
Hampshire College, Amherst, MA 01002

Converting documents to knowledge bases requires that the computer function as an intelligent document "reader" or "viewer." This artificial intelligence task involves computer vision and natural-language understanding. The Biological Knowledge Laboratory at Northeastern University is developing such a system. The lab's goal is to develop a knowledge base of biological research papers that supports the *Scientist's Assistant*, an intelligent system that will provide a scientist with interactive access to the research results, methods, and reasoning in a collection of scientific papers.

Document Understanding System. The system (see Figure 1) comprises a series of modules, starting with document scanning and ending with the Scientist's Assistant. The Scientist's Assistant is based on the paradigm of conceptual retrieval, which allows the user to find specific passages and data even if the user doesn't know the exact form in which the material is stored. With the Scientist's Assistant, a scientist ultimately should be able to point to a feature in a diagram from an older paper and ask, "Do people now understand the origin of this?" The Scientist's Assistant will then be able to find the most recent discussion of the phenomenon. For this to occur, the diagrams in the documents will have to be analyzed and the diagram contents added to the knowledge base.

We already know how to implement some of the modules shown in Figure 1, but implementing other modules will require extensive research and experimentation.[1,2] This means that it is not yet possible to test the entire system or even get input for some of the later modules when they depend on modules still under development. To avoid this impasse, we use alternative paths through the system during development and testing (represented by the dashed lines in Figure 1).

The novel aspects of the system include the design and use of *graphics constraint grammars* for describing and analyzing diagrams, the use of spatial indexing in diagram analysis and understanding, and extensions of natural-

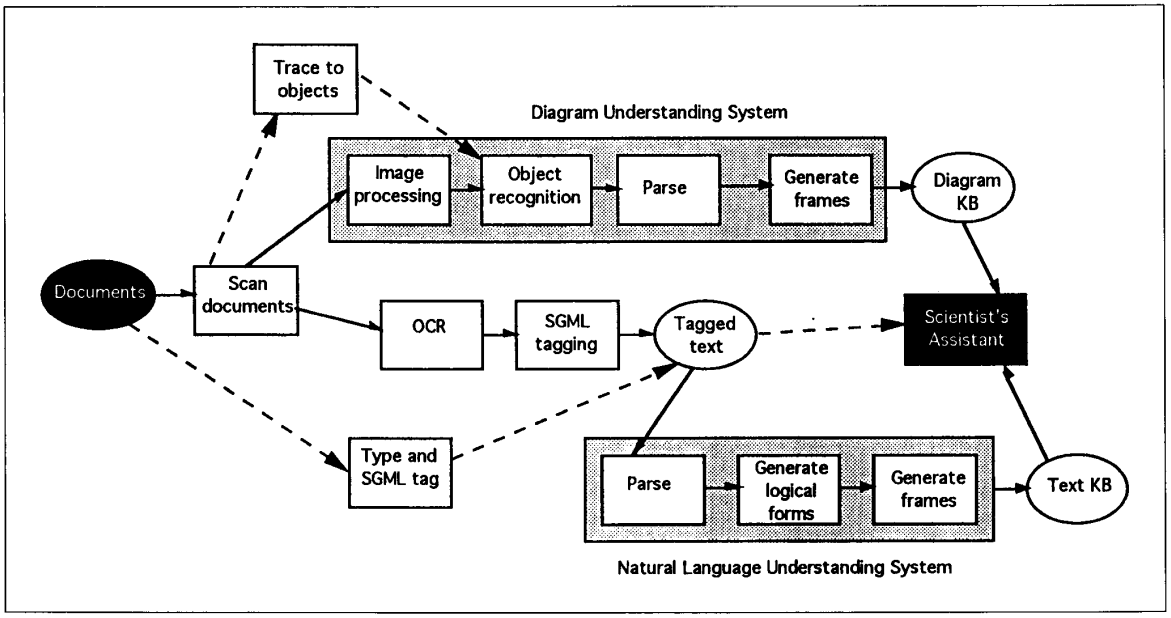

Figure 1. Overview of the Document Understanding System. Ovals denote databases and knowledge bases, rectangles represent processing systems and subsystems, and dashed arrows show the alternative strategies used during system development.

language processing techniques to complex scientific text. This article will emphasize the Diagram Understanding System. See Futrelle et al.[1] for a discussion of text processing.

The Diagram Understanding System (shown in Figure 1) entails:

Documents: Our corpus contains 1,518 papers covering essentially all of a subfield of biology (bacterial chemotaxis) from its beginning in 1965.

Object form of diagrams: Using image processing or the alternative of tracing over scanned images, the diagrams are converted from scanned, pixel-based images into a collection of graphical objects, such as lines, polygons, and positioned text.

Diagram understanding: Graphics constraint grammars are used for syntactic and semantic diagram analysis. Spatial indexing is used to rapidly discover spatial relations. The output is knowledge frames.

Tagged text: This comprises an indexed database of the entire text of each paper, encoded using the Standard Generalized Markup Language (SGML). The encoding marks every logical element such as sections, paragraphs, and sentences, as well as notations such as superscripts, subscripts, and Greek letters.

Natural-language understanding: This is a complex enterprise employing lexicons, grammars, parsers, and semantic interpreters, resulting in linked knowledge frames representing text semantics.

The Scientist's Assistant, the intelligent system that allows a scientist to navigate through the knowledge bases, is the goal of the project.

Diagram Understanding System — graphics constraint grammars. Just as we must learn the language we read and write, we must also learn the representational conventions of diagrams. Once learned, the process of reading or interpreting a diagram is relatively effortless. However, for a machine, these tasks are not so easy. A human must first describe the representational conventions formally and concisely in a way that allows the computer to carry out an analysis. For diagrams, this description is similar to the grammars that are written to describe natural language.

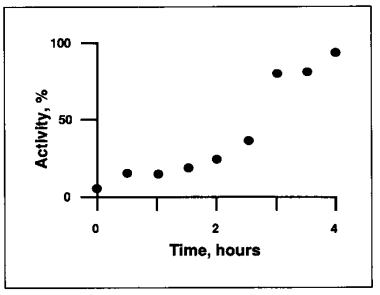

Figure 2. A data graph is the common type of diagram appearing in scientific and technical papers. The data points are the most important informational elements. The more regularly arranged elements, such as the tick marks, serve a supporting role.

One of the major conventions in diagrams is the separation of the informational components from the substrate on which the information is presented. For example, in Figure 2, the primary informational items are the data points. The vertical and horizontal scale lines are substrate items, serving as a "frame" in which to present the data. This division of labor can be subtle. For example, the positions of the circular data points are informational, whereas the diameters of the circles are not.

We describe the organization of diagrams with grammars. A grammar is a logical specification of a possibly infinite set of structures. It specifies a set of objects, the objects' attributes, and their relations. In the graphics constraint grammars we have developed, low-level elements are objects such as lines and polygons. High-level objects are more complex structures such as Data_points or Scale_lines. Graphics constraint grammars are similar to the approach Helm, Marriott, and Odersky[3] developed independently. The major difference in the approaches is that ours includes generalized equivalence relations and spatial indexing (both described below). Another difference between the approaches is that our grammars are incorporated in a complete system for document understanding.

Each graphics constraint grammar is a collection of rules (see Figure 3) comprising a production, a set of constraints, and a set of propagators:

- A production names the rule object as its left-hand side and the constituents of the object as its right-hand side.
- Constraints consist of spatial relations (such as Near, Horizontal, Aligned, etc.) as well as type constraints, which require that an object be of a certain type, such as a line or text.
- Propagators describe the relations between the attributes of the rule object and the attributes of the constituents. For example, the Center attribute of a set of lines might be computed as the center-of-mass of the set of lines.

The constituents of a rule may each be complex entities defined by still other rules. This allows us to build hierarchical descriptions of complex diagrams.

Example diagram and grammar. The example diagram of Figure 2 is a typical data graph. The highest level object of its hierarchical description has the constituents Data_set, Vertical_scale, and Horizontal_scale. The Horizontal_scale in turn has constituents Horizontal_scale_line and Axis_label—"Time (in hours)" in the example.

Figure 3 presents a fragment of the graphics constraint grammar for a data graph. Rule 1 defines the rule object, Horizontal_scale_line, with constituents Horizontal_axis_line and Ticks. The Ticks object is of type Labeled_x_ticks, which rule 2 defines as a pair of sets whose elements are Tick and Label, respectively. The propagator for rule 1 sets the value of the Head attribute of Horizontal_scale_line to the line object, Horizontal_axis_line. The Head attribute is the single item that best represents the rule object. The propagator for rule 2 sets the value of Head to a bounding box, the smallest rectangle surrounding all the objects in the sets Tick_marks and Label_set.

Generalized equivalence relations as constraints. One of the distinguishing

characteristics of the substrate of the data graph in Figure 2 is its simple and regular organization. For example, the x-axis tick marks are horizontally aligned and equally spaced. This organization is reflected in the two corresponding constraints in rule 2.

Sets of items that group together like the tick marks in the example can be described by equivalence relations. A simple equivalence relation is Equal_length. When applied to a collection of lines, Equal_length divides the lines into a collection of nonoverlapping equivalence classes, each containing lines of the same length. An equivalence relation is reflexive, symmetric, and transitive. We have extended the notion of the equivalence relation to that of the generalized equivalence relation. A generalized equivalence relation generalizes an ordinary equivalence relation in two ways:

(1) It can be approximate in nature so that it can produce classes that overlap.
(2) It has grouping relations (such as Equal-spaced) that are not normally thought of as equivalence relations.

An example of a strict equivalence relation is Coincident, referring to the positions of two objects. A generalization of Coincident is Near, a relation of great importance in diagram analysis. If two objects are near one another, they often have a logical relation, as do tick marks and their labels. Near is a generalized equivalence relation; it is not a true equivalence relation because it violates transitivity (for example, if A is Near B and B is Near C, it is not necessarily true that A is Near C). Another useful generalized equivalence relation is Strictly_near used in rule 1. Strictly_near requires that all parts of one object be near some part of another; it is not a symmetric relation.

Rule 1:

Production:
Horizontal_scale_line ⇒ Horizontal_axis_line, Ticks

Type constraints:
(Line Horizontal_axis_line)
(Labeled_x_ticks Ticks)

Geometrical constraints:
(Strictly_near Horizontal_axis_line Ticks L1)

Propagators:
Head ⇐ Horizontal_axis_line

Rule 2:

Production:
Labeled_x_ticks ⇒ Tick_marks, Label_set

Type constraints:
(Set_and_members Tick_marks Tick)
(Set_and_members Label_set Label)
(Line Tick)
(Text Label)

Geometrical constraints:
(Vertical Tick)
(Horizontally_aligned Tick_marks)
(Equal_spaced Tick_marks)
(Vertically_aligned :some Tick :every Label)
(Near :some Tick :every Label L2)

Propagators:
Head ⇐ (Bounding_box Tick_marks Label_set)

Figure 3. Two graphics constraint grammar rules that describe the horizontal scale line in the data graph shown in Figure 2. Rule 1 refers to an object of type *Labeled_x_ticks*, which is in turn defined by rule 2.

Efficient parsing of graphics constraint grammars. Solving a graphics constraint grammar problem can be an expensive computation. In finding a solution to a given rule, a number of possible assignments of objects to variables might have to be tried. This is the classic constraint satisfaction problem.[4] The usual combinatorial explosion met in these problems is mitigated by adopting the hierarchical view, which factors the problem into a set of small, independent problems that can be solved sequentially. Furthermore, the constraints that deal with the largest number of objects are typically generalized equivalence relations. These are designed to generate only solutions that include the maximal number of objects satisfying the constraint. In this way, large numbers of elements, such as the data points or tick marks in data graphs, are turned into single entities before they have to be dealt with in higher level rules.

Another potentially expensive set of computations involves geometrical relations such as Near or Aligned. For example, given an object A, we might need to find all objects B within a distance L from A, that is, satisfying (Near A B L). The normal method of doing this is to inspect every object in the diagram, compute its distance from A, and compare that to L.

Given the large amount of random-

access memory in modern machines, it is more efficient to do such computations by precomputing large data structures that make such computations run quickly — that is, by trading space for time.

In the Diagram Understanding System, this is done by building a pyramidal data structure.[2] Each level of the pyramid is a square array of cells representing the diagram at a different level of resolution. During the precomputation, each graphic object is examined, and a reference to the object is placed in any cell touched by or containing the object. The pyramidal data structure then operates as a spatial index.

Given a point in space, the objects at that point can be found immediately. Conversely, given any object, the cells it occupies are immediately available in a list stored in the object. The approach is general, because the same cell-based representation is used whether the objects are lines, polygons, curves, or text. Therefore, only one version of each geometrical constraint algorithm needs to be written — one that deals with cells.

Spatial indexing can then be used to efficiently compute a constraint such as (Near A B L). A level of the pyramid is picked on the basis of the parameter L. Only the cells adjacent to the cells including A are examined, and all the objects found in those cells are returned. The resolution of the pyramid stops well short of pixel-level resolution, so the pyramidal data structure is not particularly large — typically no larger than 128×128.

Some objects cover a lot of area, so it is inefficient to generate the number of cell references required for them. For example, as the parsing proceeds, bounding boxes to high-level objects, such as the one propagated in rule 2, might be quite large. They are stored in a different data structure, optimized for the efficient computation of constraints.

There are many complex issues in diagram parsing that we will not attempt to discuss here. For example, a diagram might have many interpretations, so information in a figure caption could be used to narrow the interpretation. Also, once the data is extracted from a data graph, further analysis is necessary to find data maxima, regions of high slope, etc., for building the knowledge representation that is to be queried.

Results. Thus far, the text of 137 articles has been encoded using SGML, and 270 diagrams have been converted to object form. The Diagram Understanding System is working; about a dozen diagrams have been analyzed with early versions of the system. For data graphs, the analysis has been able to reconstruct the datapoint values themselves. The current prototype of the Scientist's Assistant incorporates object-based diagrams — not bitmaps — and contains automatically generated hypertext links between text references and figures, tables, bibliographic items, and footnotes. Biologists have used the prototype system and given us valuable feedback. This feedback is helping to guide the ongoing task of incorporating more knowledge-based features in the assistant.

Conclusions. The technology we are developing has countless applications. The biomedical literature alone has grown by 7 million items since 1966 (as indexed in the Medline on-line information retrieval service) and is increasing at a rate of 300,000 items per year. Essentially every one of the 7 million items is available solely in hard copy, so all the techniques described here are necessary if any of this knowledge is to be converted into electronic form.

In the future, when scientific "papers" are originated, stored, and accessed purely electronically, it will still be necessary to analyze the text and diagrams in these electronic documents in order to build useful knowledge bases. The research described here is helping to prepare us for the age of fully electronic documents. ■

Acknowledgments

We thank Kent Wittenberg for insight provided during relevant discussions. This research was supported in part by National Science Foundation Grant No. DIR-8814522.

References

1. R.P. Futrelle et al., "Preprocessing and Lexicon Design for Parsing Technical Text," *Proc. Second Int'l Workshop Parsing Technologies*, Assoc. Computational Linguistics, Morristown, N.J., 1991, pp. 31-40.

2. R.P. Futrelle, "Strategies for Diagram Understanding: Generalized Equivalence, Object/Spatial Data Pyramids, and Animate Vision," *Proc. 10th ICPR, Int'l Conf. Pattern Recognition*, Vol. 1, IEEE CS Press, Los Alamitos, Calif., Order No. 2062, 1990, pp. 403-408.

3. R. Helm, K. Marriott, and M. Odersky, "Building Visual Language Parsers," *CHI 91, Conf. Human Factors in Computing*, ACM, New York, 1991, pp. 105-112.

4. J.A. Mulder, A.K. Mackworth, and W.S. Havens, "Knowledge Structuring and Constraint Satisfaction: The Mapsee Approach," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 10, No. 6, Nov. 1988, pp. 866-879.

Robert P. Futrelle is associate professor of computer science and head of the Biological Knowledge Laboratory at Northeastern University. From 1975 to 1985, he served on the biology faculty of the University of Illinois at Urbana-Champaign.

Ioannis A. Kakadiaris, who received his MS in computer science from Northeastern University, is a computer science PhD student at the University of Pennsylvania. His research interests include diagram understanding, foveal sensing, and parallel processing.

Jeff Alexander is a graduate student in computer science at Northeastern University, where he received his MS. He also works at the Charles Stark Draper Laboratory in Cambridge, Massachusetts.

Catherine M. Carriero is a graduate student in computer science at Northeastern University. Her research interests include diagram understanding and parallel algorithms.

Nikos Nikolakis is a PhD student in computer science at Northeastern University. His interests include image processing and diagram understanding.

Joseph M. Futrelle has worked for the University of Illinois and Symbolics Inc. He is an undergraduate at Hampshire College, Amherst, Massachusetts, concentrating in computer music.