Strategies for Diagram Understanding: Generalized Equivalence, Spatial / Object Pyramids and Animate Vision

Robert P. Futrelle*

Biological Knowledge Laboratory, College of Computer Science, 161CN
Northeastern University, 360 Huntington Avenue, Boston, MA 02115
futrelle@corwin.ccs.northeastern.edu

Abstract

Virtually all scientific documents contain informational diagrams. To build knowledge bases representing this literature, the diagrams must be semantically analyzed. In our system, diagrams are represented as collections of geometric objects. The fundamental organizing principle for these objects is the Generalized Equivalence Relation (GER). Examples include *Near, Parallel,* and *Aligned*. These relations can be computed efficiently using GOSSAMER, a pyramidal data structure that allows spatially associative access to objects. Animate Vision (AV), in which the image is scanned either continuously or discontinuously, is used to mimic the efficient strategy used by humans to view diagrams. The system is implemented in Common Lisp and is being applied to data graphs and gene diagrams in the biological literature.

1. Introduction

Ideas that are difficult to explain in words are usually illustrated through *informational diagrams* [1]. These include data graphs, schematic drawings of structures from molecules to galaxies, and representations of abstract entities such as processes or family trees [2]. To be optimally accessible, the electronic literature of the future will have to be encoded as a *knowledge base* after semantic analysis of both the text *and* the diagrams.

It is the analysis of diagrams that is our concern here. For example, a data graph such as Figure 1 can be analyzed in stages: Vectorization and OCR would be applied to produce geometric and text objects. Next it would be analyzed at the *syntactic level* -- the scale lines and their labels and numerical values would be found, the data points would be identified and their values determined in relations to the scales. Then semantic analysis would be done to capture concepts such as, "the data values are monotonically increasing", and other statistical information. The ultimate semantic analysis would merge this with the accompanying text which might indicate that the response in Figure 1 was "more rapid for chemical X than chemical Y", etc.

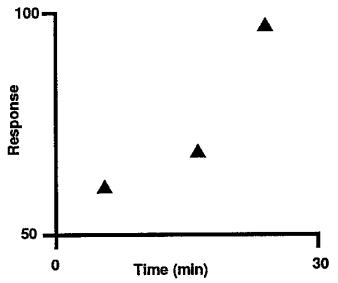

Figure 1. A simple data graph diagram. Some of the most salient features are the use of vertical and horizontal alignment and repeated elements.

Though there is ongoing research on vectorization and optical character recognition, much less is done on the syntactic and semantic stages. We will focus on these later stages by assuming that the diagrams are already available as flat files of objects, e.g., as the 21 objects labeled in Figure 2. This paper describes the concepts and techniques we have developed that will support syntactic analysis.

This paper focuses on three new ideas in computer vision that we have developed and implemented for the diagram understanding problem. The first is a dual multiresolution (pyramid) data structure, GOSSAMER, that acts as a *spatially associative database* for graphical objects. The second is the organization of objects in the image by *generalized equivalence relations, GERs*. These are implemented as discovery procedures that return corresponding static relational data objects that can then be used for pattern matching.

* This work was supported in part by grant DIR-88-14522 from the Biological and Behavioral Sciences Directorate of the National Science Foundation.

The third idea is the use of *animate vision*, AV, for static scenes. AV is the analog of human visual scanning of a scene. Our techniques are robust and efficient and have implications for other areas of image processing and computer vision.

The system described here is implemented in Procyon Common Lisp and CLOS (the Common Lisp Object System) running on 5 to 8MB Macintosh IIs and will be moved to Symbolics machines with the release of version 8.0 of their Genera system that includes CLOS.

2. Diagram structure and GERs

Informational diagrams contain two classes of information which can be called *background* and *foreground*. For Figure 1, the background consists of the x and y scale structures and the foreground consists of the three data points. The spatial organization of background material is simple and contains redundant elements. Foreground material consists of the information-bearing data elements, which by their very nature are rarely redundant or simply organized.

Graphical elements that form a single higher-level structure are typically related by what we have named *generalized equivalence relations (GERs)*. For example, one way to indicate an association between two objects is to place them *near* one another, such as the numbers and their corresponding tick marks in Figure 1. *Near* is a generalized (or approximate) equivalence relation. (*Near* is not a strict equivalence relation because it is not transitive as the stricter *coincident* relation is.) Other GERs include *aligned, parallel, same_shape, equal_length, same_type* and more specialized relations such as *strictly_near, vertically_aligned* and *horizontally_aligned.*

Figure 1 abounds with examples of GERs. The "50" and "100" are *vertically_aligned*; they are also of the *same_type*, integer. The tick marks at 50 and 100 are *vertically_aligned* also and the 50 and 100 are each *horizontally_aligned* with and *near* their respective tick marks.

The basic *near* relation is of the form,

Near(ob1, how_near, ob_set)

such that given a geometrical object *ob1*, all objects in the set *ob_set* have a minimal separation from *ob1* that is less than or equal to *how_near*. If the procedure is given *ob1* and *how_near*, it returns a data object, a static *near* object, containing the three items, *ob1, how_near, ob_set.*

Many GERs are related to *near*. These GERs apply to all four classes of geometric objects shown in Figure 2. (The points do not appear explicitly in Figure 1 but are generated as *terminators* of the lines.) In Figure 2, the top tick mark L1 is *strictly_near* the long vertical line L2, but the converse is not true. To find what items are *horizontally_aligned* with the top tick mark L1, a horizontal line is constructed (not shown) through the tick mark and all items that are *strictly_near* to the constructed line are considered *horizontally_aligned* with L1. This means for example that L2, which is not *strictly_near* L1, will not be counted as *horizontally_aligned* with L1.

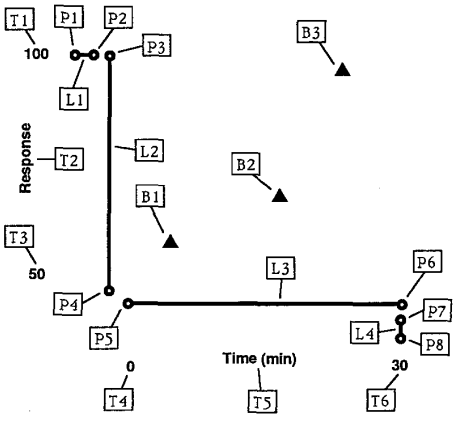

Figure 2. The diagram of Figure 1 is shown exploded with labels for the geometric objects -- L for lines, T for text, P for points, and B for blobs. The terminators of lines are so important that they are introduced as separate geometric objects (points).

3. The GOSSAMER data structure

We will give the motivation for the design of GOSSAMER and then describe its implementation. Since the *near* GER is basic, a data structure that allows it to be computed quickly is good choice. If we look at the long vertical line in Figure 1, it is easy to notice items that are near the line. We simply scan the line with our eyes and the tick marks and numbers are seen close to the line in our visual field. We would like a data structure that has this same simple property: if we use a point in space as the index into the structure then the value returned is the set of objects that have parts near the point. This simple idea is the basis of GOSSAMER. The GOSSAMER data structure is a square array of square cells in which each cell that has any part of a geometric object in it has a reference to the object. GOSSAMER is actually a multiresolution collection of such arrays with additional structure to be described.

There are a few immediate objections to this approach that need to be addressed. First, the data structure would appear to be enormous, because we might use a resolution of 1024 x 1024 so that each line or text object mighthave to be referenced in thousands of cells. The answer is that the data structure is designed to organize

geometrical *objects*; it does not need resolution at the pixel level. A 128 x 128 (16K) array gives resolution to the single character level in a typical diagram. Currently, 8MB Macintoshes easily support 128 x 128 arrays with hundreds of thousands of pointers to objects even without virtual memory. Another problem is that when there are multiple references to objects they will have to be processed repeatedly. This objection has some validity but we use tagging and recursive algorithms that limit the complexity of calculations to linear or logarithmic in the object size (number of cells occupied) or the number of objects in a region. Some computations take only constant time, e.g., to find all objects near a point requires looking only at the point's cell and its eight neighbor cells.

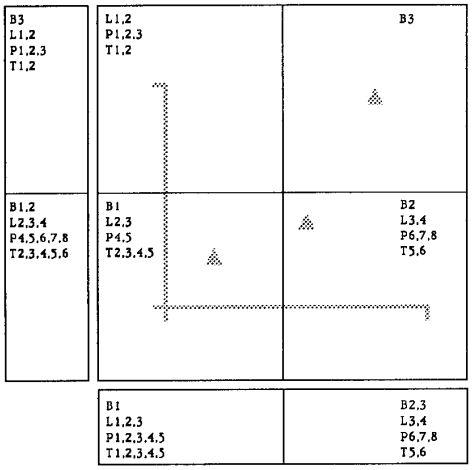

Figure 3. Representation of Figure 1 in level 1 of the GOSSAMER pyramidal data structure. (The greyed objects are to orient the reader.) Each square cell contains pointers to all geometric objects that touch it, where the pointer labels were given in Figure 2. The x and y projection arrays are shown at the bottom and left, respectively. The full GOSSAMER structure is a pyramid extending to $n=7$, typically. This is a *spatially associative structure*, because querying any small region of space immediately returns the set of all objects touching that region.

3.1 Implementation of GOSSAMER

GOSSAMER is implemented as a pyramid, a multi-resolution collection of arrays with $2^n \times 2^n$ cells in each layer, $n=0,\ldots,nmax$, where typically, $nmax=7$. Each layer of the pyramid represents the total x,y space, at one resolution. The pyramid uses 4/3 the memory space that the bottom (highest resolution) level uses. Each element of the array has a single *cell* object as its value. The most important slot in a cell is *content*, the list of geometrical objects it contains. Each cell is linked to its four children and to its parent, forming a complete quadtree. The GOSSAMER structure combines the M-pyramid (matrix-based) and T-pyramid (tree structure) [3]. To rapidly analyze horizontal and vertical alignment, two one-dimensional arrays exist at each level whose cells contain the union of all cell contents in the corresponding row or column.

Figure 3 shows level 1 of the pyramid corresponding to the items in Figure 2. Each geometrical object is *installed* in the GOSSAMER structure by adding it to the contents of the appropriate cells at the lowest, *nmax*, level of the pyramid and propagating the contents up the tree by set union. There are specialized methods for the installation of each class of geometrical object. All geometrical objects and cells have a *tag* slot that allows operations such as set union to run in linear time. For example, when a parent cell has its *content* slot filled from its children, a unique tag is generated for the parent, each object added is tagged, and no tagged object is added to that parent again. Tagging is especially efficient near the top of the pyramid because otherwise, lists of hundreds or thousands of objects would have to be searched to avoid or remove duplicates.

3.2 How GOSSAMER is used to compute GERs

When a geometrical object is installed in the pyramid the cells it occupies at each level are recorded within the object. In order that *near* and similar relations can be computed correctly, avoiding near misses, the set of 8-neighbors of the occupied cells is also recorded. These sets of cells become the primary means of characterizing geometrical objects and computing the various GERs. This makes it possible for example, to treat lines, curves, and text on an equal footing -- all are represented by sets of cells. This means that *near* and the other GERs are each computed by a single algorithm for all classes of geometrical objects.

Referring to Figure 4, the operation of the *near* algorithm is as follows: *Near* is called with the arguments L and *how_near*. The value of the *how_near* distance is used to select a level in the pyramid so that the objects found are, on average, no more than *how_near* from L. In Figure 4, the level $n=3$ is shown. At this level there are 9 cells occupied by L and 24 additional 8-neighbor cells. These 33 cells are examined and all geometrical objects within them, other than L itself, are collected into a set (using tagging to avoid the 2nd, 3rd and 4th occurences of line A). The result is that the circle in cell 16 and line A are returned as the set of items *near* L. The 8-neighbors are used to avoid "near misses" such as the circle.

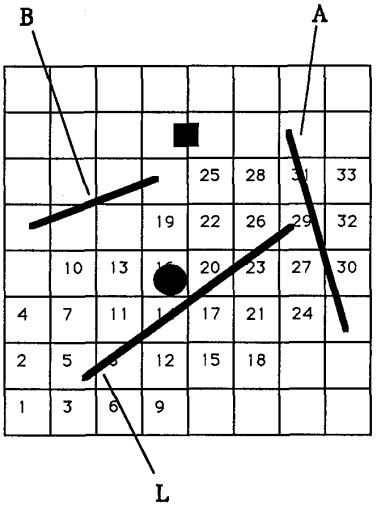

Figure 4. The operation of *near*, a GER (Generalized Equivalence Relation) at pyramid level 3. The line L passes through 9 cells and has 24 8-neighbors. Objects (A and the circle) touching these 33 cells are considered *near* to L at this resolution. In animate vision (AV), the cells are scanned in the order shown. The scan may interrupt and return when the circle is found and then be restarted later to find A.

The algorithm for computing *near* is linear in the length of the line, its first argument. The pyramid of cells allows us to implement a recursive descent algorithm which is linear in the depth of the tree, which is logarithmic in the resolution. The algorithm can be described by focusing on the discovery of the top tick mark in Figure 1 as an object that is very near to the long vertical line, see Figure 5. The algorithm works as follows:

1. Begin at the top of the pyramid and put the root cell into *cell_list*.

2. Retain from *cell_list* only those cells that contain both the object and possible neighbor objects.

3. If the level of the pyramid corresponding to *how_near* is reached, return the current other objects.

4. Else, replace *cell_list* with the children of each cell in *cell_list* and go to 2.

Figure 5 shows the cells retained at step 2 for pyramid levels 2 through 7. Referring to Figure 1, we see that this algorithm would converge on *two* objects, the top tick mark and the long horizontal line. This requires the examination of about 9 cells per level for each of the two neighbor objects, about 100 cells in total. The linear algorithm that looks at all 8-neighbors of the line at the $n=7$ level has to examine about 300 cells to do the same computation. The main reason the recursive algorithm is not much faster is the modest depth of the tree.

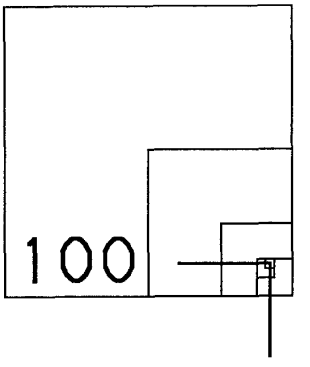

Figure 5. Schematic of the recursive descent procedure to find the tick mark very near to the top of the vertical line in Figure 1. At each pyramid level only the subset of neighbors that contains *both* the line and any possible neighbors is retained. The cells retained from levels 2 through 7 are shown.

3.3 Representation of regions

It is important to be able to represent large regions efficiently in GOSSAMER. They cannot be represented easily by the methods outlined so far because a single region might be referenced in 10,000 cells. Instead, regions are treated as special geometric objects and represented as quadtrees made up of square cells drawn from different levels of the pyramid and integrated into the GOSSAMER structure.

Efficient algorithms to compute functions on quadtrees such as point containment, union and intersection are exhaustively discussed by Samet [4]. Some of these methods can be further speeded up by tagging. Since the 8-neighborhood of a quadtree is itself a quadtree, near misses can be avoided. Quadtrees are intrinsically multi-level, so additional installation processing has to be done to represent them at different levels of resolution.

3.4 "Aspects" of a diagram

We have described the use of constructed lines in computing GERs such as *horizontally_aligned*. These constructed lines must not be confounded with the original *source* diagram. For this reason we have

made it possible to define an arbitrary set of geometric objects as an *aspect*. The two primary aspects we use are **source** and **constructed**. It can be useful to divide the **source** further into the**to_do** and **done** aspects. Initially all of the source is in **to_do** aspect, and then as each collection of objects is analyzed and accounted for the objects are moved into **done**. In keeping with our introductory discussion, it can also be useful to have **foreground** and **background** aspects. All GER methods take an aspect as an (optional) argument so the filtering of the *ob_set* is done within the procedure, not afterwards.

4. Animate Vision (AV)

Human vision has a focal region of high acuity a few degrees wide in the center of the field where the *gaze* is directed. Outside of this, *peripheral vision* gives a low-resolution image. To gather information about a complex diagram, a person scans the diagram in a series of successive *fixations* of the gaze. The gaze is often moved along prominent lines, or if there is a gap in the figure, the gaze is moved discontinuously to some new region where there are objects to examine [5]. The decisions for gaze movement are made on the basis of analysis goals, information gathered so far, and information from peripheral vision. Ballard has called these aspects of vision, *animate vision* [6].

4.1 Peripheral vision in GOSSAMER

One of the motivations behind the design of GOSSAMER was to make it possible to efficiently implement animate vision. Peripheral vision is modeled by developing statistical summaries of large regions at low resolution, information collected in the large cells near the top of the pyramid. Various statistics are computed at the highest level of resolution and then propagated up the pyramid by summation or averaging [3,7]. For example, the total number of objects in a region can be summed, or the estimated "blackness" can be averaged (based on line length times line width within each cell).

4.2 Scanning algorithms

The *near* relation directly implements a form of peripheral vision in which other objects are "visible" if they are near the object in question. A scanning version of *near* has been implemented in which a *scan_state* object is defined and given to a scanning procedure. The procedure inspects cells sequentially and can interrupt and return an updated *scan_state*. The situations for which an interrupt can be requested can be the discovery of a particular type of object, e.g., a short vertical line, or a reaction to any unusual object not on an "expected" list. Scanning gives the system a multitasking capability because one scan can be suspended while another is initiated to follow a promising lead. To avoid conflicts, object tagging for scanning is done on a per-process basis. An example of scanning is given in Figure 4.

5. Strategies for analysis

It is not the pupose of this paper to go into actual diagram analysis methods, which are still under development. But a few remarks are in order. Since the GERs return static relational objects, it is possible to do various types of pattern matching at higher levels. We hope to avoid matching to detailed and specific models because we think that generalized equivalence should be able to capture many of the regularities of diagrams independently of detailed models. There is also a lot to be gained by studying the successful but distinct approaches to the analysis of graphics by Srihari's and Kasturi's groups [8,9]

One of the attractive control paradigms is the blackboard architecture, because through the **constructed** aspect and other aspects we can delineate regions to explore and maintain multiple hypotheses as needed -- in a sense the two-dimensional space can act as its own blackboard. Another area we are exploring is the use of numerical or fuzzy measures for GERs, e.g., if two lines are not the same length, how close in length *are* they?

6. Discussion

About 30% of the space of scientific papers is not text, but diagrams, photos, and tables [1]. So we work on encoding both the text *and* graphics in documents. This is the emphasis of our Biological Knowledge Laboratory at Northeastern [10,11].

One of our goals is to produce a computational analog of human gestalt vision, or *Pragnanz* [12]. In vision a great deal of organization is discovered preattentively [13]. These rapidly discoverable items are the ones that can be rapidly grouped into equivalence classes (our GERs), such as X's on a background of O's.

GERs are closely related to optimization and regularization techniques [14]. For example, *vertically_aligned* represents a minimal cost solution to a constrained geometric problem. Our methods can implement the notion of "fuzzy equality", similar to GERs, discussed by Witkin and Tenenbaum [15].

Arguments for animate vision have been given in [6], but they are focused on moving objects or cameras. But scanning static images is an integral part of human vision. Yarbus [16] has shown that the particular task given to a subject influences the patterns of scanning. Scanning is also related to Ullman's "visual routines" [17], though he hardly discusses decision strategies.

He does discuss topological issues such as *open/closed, interior/exterior,* and *connected/not_connected.* GOSSAMER is well suited for topological computations because it can do them above the pixel level to get results rapidly.

We have already discussed issues of occlusion, when one diagram element partially overlays another [18]. Another important problem is inherent in our technique, the problem of choosing scale lengths for GERs such as *Near.* We are looking at clustering techniques to do this. Approximate geometrical relations has been developed in Davis's thesis [19].

In the future, electronic documents will be stored in a standard form such as SGML [20] and the diagrams in another standard such as Postscript [21]. Data will be stored in a spreadsheet format which would allow the reader to choose the display format. Even then, the approaches we've described will be useful. There are millions of articles that exist only as hardcopy and need to be converted to electronic form. We are doing this for a subset of the biological literature [10,11]. Also, it will be a long time before all of the major sources of technical literature will be produced and distributed in electronic form. Many diagrams will only be available in a low-level form and will still require syntactic and semantic analysis. Even spreadsheet data will require semantic analysis before a knowledge base can be built.

Acknowledgements

Thanks to all the students who have worked on the Diagram Understanding project to date: David R. Reed, Bob Sapienza, Michelle Guardabascio, David Taylor, Georgios Evangelidis, Paul Steckler and Sougata Mukherjea. A dicussion with J. Shah on theoretical issues was useful. Thanks to M. Pescitelli, E. Nicholson, Y. Kakadiaris and D. S. Ellis for editing help.

References

[1] Futrelle, R. P. (in press) The Skyline of Graphical Information. *In: AI: Expert Systems and Other Applications; Selected Papers.* Smith, L. C. (Editor) Greenwood Press, Westport, Connecticut.

[2] Tufte, E. R. (1983) *The Visual Display of Quantitative Information.* Graphics Press, Cheshire, Connecticut.

[3] Tanimoto, S. and Klinger, A., (Editors) (1980) *Stuctured Computer Vision.* Academic Press, NY.

[4] Samet, H. (1990) *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, MA.

[5] Intraub, H. (1981) Identification and Processing of Briefly Glimpsed Visual Scenes. *In: Eye Movements: Cognition and Visual Perception.* Fisher, D. F., Monty, R. A., and Senders, J. W. (Editors) Lawrence Erlbaum, Hillsdale, NJ. 181-190.

[6] Ballard, D. H. (1989) Reference Frames for Animate Vision. IJCAI-89, 1635-1641.

[7] Shneier, M. (1984) Multiresolution Feature Encodings. *In: Multiresolution Image Processing and Analysis.* Rosenfeld, A. (Editor) Springer-Verlag, Berlin, 190-199.

[8] Kasturi, R., Bow, S., Gattiker, J., Shah, J., El-Masri, W., Mokate, U., and Honnenahalli, S. (1988) A System for Recognition and Description of Graphics. *Proceedings of the 9th International Conference on Pattern Recognition.* IEEE Computer Society, 255-259.

[9] Wang, C., and Srihari, S. N. (1988) A Framework for Object Recognition in a Visually Complex Environment and its Application to Locating Address Blocks on Mail Pieces. *International Journal of Computer Vision,* **2**, 125-151.

[10] Futrelle, R. P. (1989) An Introduction to the Biological Knowledge Laboratory. *Technical Report of the College of Computer Science, Northeastern University.* NU-CCS-89-15.

[11] Futrelle, R. P. and Pescitelli, M. J., Jr., (in press) The Scientist's Assistant: Building Biological Knowledge Bases. *Computer Applications in the Biosciences.*

[12] Pomerantz, J. R. and Kubovy, M. (1983) Perceptual Organization: An Overview. *In: Perceptual Organization.* Kubovy, M. and Pomerantz, J. R. (Editors) Lawrence Erlbaum, Hillsdale, N. J., 423-456.

[13] Treisman, A. (1983) Preattentive Processing in Vision. *In: Human and Machine Vision II.* Rosenfeld, A. (Editor) Academic Press, New York. 313-334.

[14] Poggio, T., Torre, V., and Koch, C. (1985) Computational Vision and Regularization Theory. *Nature,* **317**, 314-319.

[15] Witkin, A. P., and Tenenbaum, J. M. (1983) On the Role of Structure in Vision. *In: Human and Machine Vision.* Beck, J., Hope, B., and Rosenfeld, A. (Editors) Academic Press, New York. 481-545.

[16] Yarbus, A. L. (1967) *Eye Movements and Vision.* Plenum Press, New York.

[17] Ullman, S. (1988) Visual Routines. *In: Visual Cognition.* Pinker, S. (Editor) The MIT Press, Cambridge, Massachusetts. 97-159.

[18] Futrelle, R. P. (1985) A Framework For Understanding Graphics In Technical Documents. *In: Expert Systems in Government Symposium,* IEEE Computer, Washington, DC 386-390.

[19] Davis, E. (1986) *Representing and Acquiring Geographic Knowledge.* Pitman Publishing Limited, Pitman, London.

[20] Bryan, Martin (1988) *SGML: An Author's Guide to the Standard Generalized Markup Language.* Addison-Wesley, Reading, MA.

[21] Adobe Systems (1985) *Postscript Language Reference Manual.* Addison-Wesley, Reading, MA.