

A Graph Based Data Model for Graphics Interpretation

Endre Katona

University of Szeged, H-6720 Szeged, Árpád tér 2, Hungary
katona@inf.u-szeged.hu

Abstract. A universal data model, named DG, is introduced to handle vectorized data uniformly during the whole recognition process. The model supports low level graph algorithms as well as higher level processing. To improve algorithmic efficiency, spatial indexing can be applied. Implementation aspects are discussed as well. An earlier version of the DG model has been applied for interpretation of Hungarian cadastral maps. Although this paper gives examples of map interpretation, our concept can be extended to other fields of graphics recognition.

Keywords: map interpretation, graphics recognition, graph based data model, vectorization, spatial indexing.

1 Introduction

Although a lot of papers have been published presenting graphics interpretation systems [5, 6]; most of them concentrate on algorithmic questions, and less attention is taken to data storage and handling. In this section we give an overview of data models that we have investigated when creating our own model.

It is a natural way to use some graph representation for a vectorized drawing. Lladós et al. [11] define an *attributed graph*, and after extracting minimum closed loops, a region adjacency graph is generated. This model concentrates on region matching and this fact restricts its applicability. In some sense similar approach is given in [15] defining an *attributed relational graph* where nodes represent shape primitives and edges correspond to relations between primitives. A special approach is applied in [1]: after an initial rungraph vectorization a mixed graph representation is used to ensure interface between raster and vector data.

Some interpretation systems use *relational database tables* to store geometric information of vectorized maps [2, 18]. The advantage of this approach is that commercial database management systems can be used to handle data. Although there are existing techniques to store spatial data in relational and object-relational tables [14], it is clear that the relational model is not the best choice for graphics interpretation.

Object-oriented models are more flexible than relational ones and can be used in graphics recognition [3] as well as in GIS (Geographical Information System)

approaches. Object-oriented GIS typically uses a hierarchy of spatial object types, such as defined in the *Geometry Object Model* of the Open Geospatial Consortium [14] supporting interoperability of different systems. The object-oriented concept is excellent for high level description, but it does not support low level algorithmic efficiency during recognition.

Topological models have been introduced in early GIS systems. A topological model can be regarded as a set of cross-referenced drawing elements: each element has a unique identifier (id) to offer the possibility for other objects to reference it. A characteristic example is the node-line-polygon structure of the Arc/Info data model [8]. This model ensures efficient algorithms to compute polygon areas, point-in-polygon decisions, overlay of two polygon sets, etc., but has some drawbacks when updating a graphical database [8].

It is a challenging approach in graphics recognition to use some knowledge representation scheme, such as *semantic networks*. Niemann et al. [13] give detailed description of a general semantic network model, applied both for vector-based [4] and raster-based [17] map interpretation. Such a semantic network model defines *concepts* to describe a priori knowledge, while *modified concepts* and *instances* are built up during interpretation. Hartog et al. [7] use a similar approach to process gray-level map images. Although semantic networks give a rather general approach, applied also in speech understanding and robotics [13], they do not support the low-level efficiency of algorithms like topological models do.

2 The DG (Drawing Graph) model

Our model combines graph-based and topological approaches, but object-oriented aspects and semantic networks are also taken into consideration in some sense. A preliminary version of DG has been applied for the interpretation of Hungarian cadastral maps [9].

Basic element of the model is the *DG-object*. A set Z of objects, describing the current state of interpretation, is called *DG-document*. Z consists of two disjoint subsets:

- Z_n denotes the set of *normal objects*; they are used to describe the current drawing.

- Z_s denotes the set of *sample objects*, giving an a priori knowledge description. For instance, sample objects can describe a symbol library of the map legend or vector fonts of a given language.

Each object has the structure $(id, layer, references)$ where id is the object identifier number, and $layer$ is a CAD-like attribute to classify objects. Normal objects have 0, 1, 2, etc. layer numbers, while sample objects are kept in a distinguished layer S . Layer 0 is reserved for unrecognized objects. Each object may have *references* to other objects using their id's. The set of references $R \subset Z \times Z$ form an acyclic directed graph, termed *reference graph*. Two types of references can be distinguished:

- A “contains” reference means that the current object involves the referred object as a component. $R_c \subset Z \times Z$ denotes the set of “contains” references.
- A “defined by” reference means that the current object is a transformed version of a referred sample object. Such references are mainly used to describe recognized instances of sample objects. $R_d \subset Z \times Z_s$ denotes the set of “defined by” references, and $R = R_c \cup R_d$ holds.

Denote $domain(u)$ the set of all objects v that have a reference path from u to v , and denote $scope(u)$ the set of all objects v with a reference path from v to u . Notations $domain_c(u)$ and $scope_c(u)$ mean restrictions to “contains” reference paths. For any sample object s , $domain(s) \subseteq Z_s$ is required.

The DG model contains three basic object types (instances of each may be normal or sample objects as well):

- A *NODE object* represents a point with coordinates; a NODE instance is denoted as $node(x, y)$. Normally, a NODE has no references to other objects.
- An *EDGE object* is a straight line section given by “contains” references to the endpoints. An EDGE instance is denoted as $edge(node_1, node_2)$. A “line width” attribute may be attached, if necessary.
- A *PAT (pattern) object* represents a set of arbitrary DG-objects given by “contains” references to its components. A PAT instance is denoted as $pat(obj_1, \dots, obj_n)$. Coordinates (x, y) of a *center point* may be attached to the PAT, usually giving the “center of gravity” or other characteristic point of the pattern.

At a first look, NODE and EDGE objects form a usual graph structure describing the drawing after initial vectorization. A PAT object typically contains a set of edges identifying a recognized pattern on the drawing, but PATs can be utilized also in very different ways.

The object type NODE has an important subtype, termed *TEXT*. Basically it represents a recognized inscription on the drawing, defined as a special transformation of a vector font. (Vector fonts are given as sample objects.) Generalizing this idea, a TEXT object can be used to describe a transformed instance of any other sample object, as will be shown in Section 3. A TEXT instance is denoted as $text(sample, x, y, T, string)$ where *sample* is a “defined by” reference to a sample object, x and y are coordinates of the insertion point, T is a transformation usually given by an enlargement factor and rotation angle, and *string* is an ASCII sequence of characters.

If *string* is given, then *sample* refers to a vector font $pat(letter_1, \dots, letter_m)$ where, for any i , $letter_i$ is a $pat(edge_1, edge_2, \dots)$ object defining a character shape. In this case $text(sample, x, y, T, string)$ describes a recognized inscription on the drawing.

If *string* is omitted, then *sample* refers to the description of a certain symbol. For instance, *sample* may refer to a $pat(edge_1, \dots, edge_n)$ object giving vector description of a map symbol with $(0, 0)$ coordinates as center point. In this case $text(sample, x, y, T)$ describes a recognized instance of the symbol at point (x, y) transformed according to T .

Since TEXT is a subtype of NODE, it can be applied as endpoint of an edge. In this way we can represent special symbols applied for instance at parcel corners on cadastral maps (Fig. 1).

It is easy to see that the above concept of TEXT ensures not only high flexibility, but also supports efficient displaying of the current recognition state during the whole process.

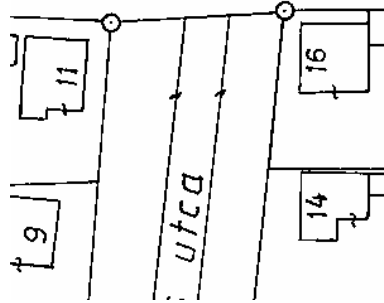


Fig. 1. Small circles denoting geodetically identified points on maps

3 Interpretation strategy

Initially the DG-document contains only sample objects coding prototypes of symbols and characters to be recognized. Interpretation starts with some raw vectorization process (see [20] for an overview of vectorization methods). As a result of the vectorization, a NODE-EDGE graph description of the drawing is inserted in the DG-document. At this moment all normal objects are in layer 0. The processing is performed as a sequence of recognition steps, each step may consist of three phases:

1. *Hypothesis generation.* PAT objects are created in the DG-document. For instance, if a set of edges e_1, \dots, e_n is recognized as a map object, then a $pat(e_1, \dots, e_n)$ is created with the layer number associated with the current map object type. Such an operation does not change the underlying data, thus the hypothesis generation is a reversible step ensuring the possibilities of backtracking and ignoring. PAT objects can describe a hierarchy of higher level structures like blocks and entities in [19].

2. *Verification of hypotheses* can be made by the user or by a higher level algorithm: PATs of false hypotheses are marked as “rejected” while correct ones as “accepted”.

3. *Finalization.* Rejected hypotheses are dropped and accepted ones are processed, possibly making irreversible changes in the underlying data. In some cases finalization can be omitted or postponed, in this way preserving the possibility of backtracking.

The above procedure will be demonstrated on two examples.

Example 1. Text recognition (Fig. 2).

1. In the hypothesis generation step small connected subgraphs are detected as candidates of single characters (see pat_1 , pat_2 and pat_3 in Fig. 2). Next, if an aligned

group of character candidates is found, a $pat(pat_1, \dots, pat_n, text_1, text_2)$ object is created where the contents of $text_1$ and $text_2$ are not defined yet. Rotation angle α of the string candidate is detected, and characters are recognized with rotation α and $180^\circ + \alpha$, respectively. Recognition results are stored in $text_1$ for α and in $text_2$ for $180^\circ + \alpha$. The recognition itself can be performed with some graph matching technique like in [11] and [12], or using a neural network [9].

2. String hypotheses can be verified by the user or applying some a priori information. For instance, when processing cadastral maps, the set of legal parcel numbers is usually given in an external database.

3. If a hypothesis is rejected, then all PATs and TEXTs, created in the hypothesis generation phase, should be deleted (Fig. 2). If accepted, then only the selected TEXT should be kept and all other objects – including the base vectors of the characters – should be deleted.

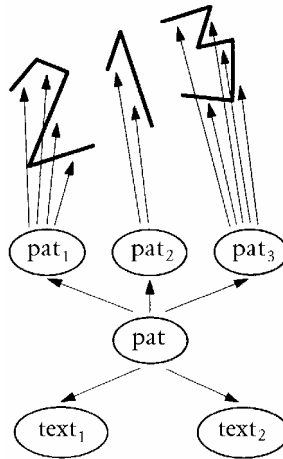


Fig. 2. Example of text recognition using the DG model. Arrows denote references between objects

Example 2. Recognition of connection signs on cadastral maps (Fig. 3). A connection sign is applied on the boundary line of two objects expressing that the two objects are logically connected (e.g. a building belongs to a given parcel, see Fig. 5).

1. In the hypothesis generation phase a $pat(pat_1, pat_2, text)$ is created to each connection sign candidate, where pat_1 involves the connection sign edges, pat_2 contains the segments of the base line (Fig. 3), and $text$ has a reference to a sample connection sign symbol.

2. Verification of hypotheses can be made by simple accept/reject answers.

3. If a hypothesis has been rejected, then pat , pat_1 , pat_2 and $text$ should be deleted. If accepted, then edges in pat_1 are deleted, edges in pat_2 are unified into a single edge e_0 , and $pat(pat_1, pat_2, text)$ is replaced with $pat(e_0, text)$ expressing the connection between the symbol $text$ and the edge e_0 . As a result, recognized connection signs are displayed correctly (Fig. 5/b).

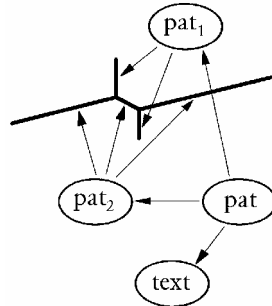


Fig. 3. Recognition of a connection sign. Arrows denote references between objects

4 Implementation

The whole DG-document can be stored in RAM, since the DG description of a whole map sheet normally does not exceed 10 Mbytes. Data structure consists of two arrays, *Obj* and *In*. The *Obj* array contains object descriptions, and *In*[*k*] gives the starting address for object with identifier *k*. (Note that description of an object does not contain its id.) This mode of storage ensures constant access time along object references.

To ensure computational efficiency, “contained in” references – as inverses of “contains” references – should be applied in some cases. For instance, all NODE objects should have “contained in” references to the connected EDGE objects, in this way efficient graph algorithms can be programmed. Note that in our implementation, when necessary, all “contained in” references can be generated in linear time.

Automatic interpretation always needs human control and corrections, therefore it is important to ensure fast displaying of the current recognition state on monitor screen. When displaying a DG-document, only straight line sections should be drawn, because all objects can be traced back to EDGE objects along “contains” and “defined by” references. Color and line style is associated with each layer number (excepting the *S* layer, because sample objects are not displayed). To demonstrate recognition hypotheses on the screen, edges in layer 0 are displayed according to the maximum layer number in their scope.

5 Spatial indexing

The above DG implementation ensures fast data access along references, but spatial searches, for instance to find the nearest node to a given node, may be very slow. The problem can be solved by spatial indexing (for an overview see [16]). There are two main types of spatial indexes: *tree structured indexes* are based on hierarchical tiling

of the space (usually quadtrees are applied), while *grid structured indexes* use homogeneous grid tiling.

Although quadtrees have nice properties in general, in the case of drawing interpretation a grid index may be a better choice, because of the following reasons. On one hand, drawing density is limited by readability constraints. As a consequence, the number of objects in a grid cell is a priori limited. On the other hand, map interpretation algorithms normally use fixed search window (for instance, when recognizing dashed lines). A grid index of cell size near to the search window size can work efficiently.

To discuss indexing techniques, we define the *minimum bounding box* (MBB) of an object as the minimum enclosing rectangle whose edges are parallel to coordinate axes. Considering the DG model, the MBB of an object z can be determined by computing minimum and maximum coordinates of nodes in $domain_c(z)$. (MBB can be defined also by $domain(z)$, in this case the MBB of a TEXT object involves not only the insertion point, but also the transformed vectors of the sample object.)

Fig. 4 shows a grid index example of 3×3 tiles where a list of object id's is created to each grid cell. An id appears in the i -th list if the MBB of the object overlaps C_i . In this way the same object id may appear is several lists.

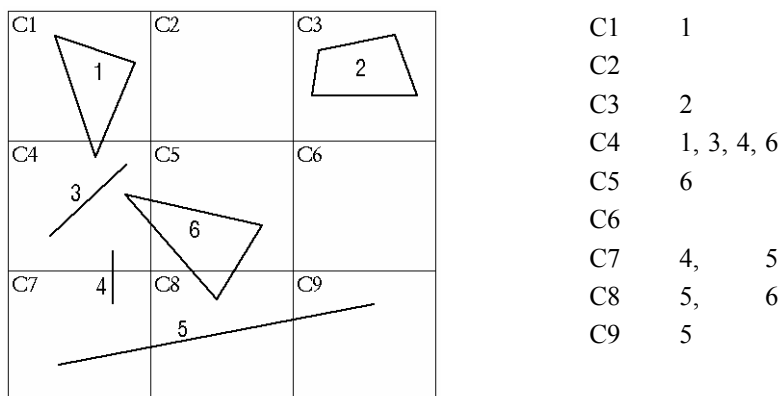


Fig. 4. Example of a grid index

Our grid index implementation [10] ensures the insertion of a new id in constant time. As a consequence, a grid index for N objects can be generated in $O(N)$ time, which is better than the usual building time $O(N \cdot \log N)$ for quadtrees.

6 Application: interpretation of cadastral maps

The DG model has been applied to interpret Hungarian cadastral maps. Main processing phases are sketched below (here we concentrate only on data modeling aspects, for algorithmic details see [9, 10]).

1. *Vectorization.* A thinning-based vectorization algorithm converts the whole scanned image into a set of vectors. Further on denote N the number of generated vectors.

2. *Creating topology.* An initial DG-document (a NODE-EDGE graph) is generated from the set of vectors. This process takes $O(N \cdot \log N)$ time (it is based on sorting the nodes according to their coordinates, and unifying nodes of identical coordinates).

3. *Dashed line recognition.* For each dashed line candidate a $pat(edge_1, \dots, edge_n)$ is created. If grid indexing is applied, then recognition can be performed in $O(N)$ time (see Section 5). If a dashed line candidate is accepted, then component edges are replaced with a single EDGE object having a layer number assigned to dashed lines.

4. *Text recognition.* String candidate PATs are created as shown in Example 1 of Section 3. A 17-element feature vector is generated for each character PAT, and recognition is performed by a feedforward neural network (for details see [9]). Although vectorized symbols may have significant distortions as compared to the raster image, the neural network can learn these distortions and can produce successful recognition (Fig. 5).

5. *Recognizing connection signs,* as explained in Example 2 of Section 3.

6. *Recognizing small circle symbols* (Fig. 1). A hypothesis $pat(edge_1, \dots, edge_n)$ is created to each small closed convex polygon. If accepted, $edge_1, \dots, edge_n$ are deleted and nodes of these edges are unified into a single TEXT object at the center point of the polygon.

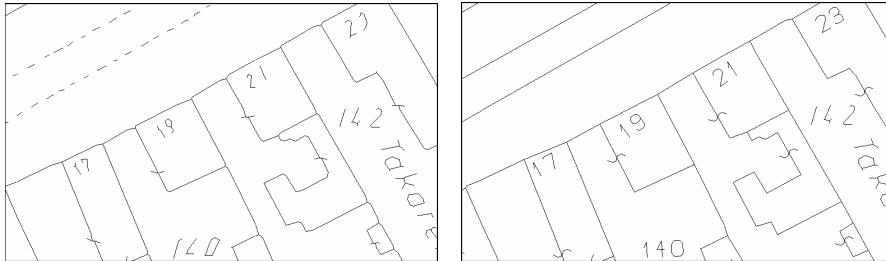


Fig. 5. Automatic interpretation of Hungarian cadastral maps: a) raw vectorized image, b) recognition result (without manual correction)

7. *Drawing correction.* The initial raw vectorization has typical anomalies at corners and T-junctions (Fig. 5/a). Each of these anomalies is recognized as a $pat(pat_1, pat_2)$ where pat_1 contains edges to be corrected and pat_2 contains the new (corrected) edges. When accepted, edges in pat_1 are deleted; otherwise edges in pat_2 should be deleted. These corrections are performed only on edges with empty scope, that is, on edges that have not been recognized till now.

8. *Recognizing buildings and parcels.* A complex algorithm is applied, utilizing recognized connection signs, house numbers and parcel numbers [9]. A PAT object is created to each building polygon and parcel polygon. If accepted, the PAT is kept, otherwise rejected.

Processing time of one recognition step takes only a few seconds for a total map sheet. This fact supports interactivity and makes it possible to create rather complex algorithms in realistic time.

7 Conclusions

A universal graph-based data model has been introduced for graphics interpretation. The same data structure is used

- to describe the original (raw vectorized) drawing,
- to describe and display the recognized drawing,
- to support the recognition process as well as manual corrections.

Our specification is independent of recognition algorithms, but suggests an interpretation methodology on one hand, and gives a technical background on the other hand.

When applying an interpretation system in practice, it is a usual difficulty that the user is not familiar with inherent algorithms and data structures, and therefore cannot make optimal control of the system. We think that basic ideas of the DG model (only with four object types) are simple enough to understand for the user, and this fact supports the efficiency of interactive work.

References

1. Boatto, L., Consorti, V., Buono, M., Zenzo, S., Eramo, V., Esposito, A., Melcarne, F., Meucci, M., Morelli, A., Mosciatti, M., Scarci, S., Tucci, M.: An Interpretation System for Land Register Maps. *Computer*, Vol. 25, No. 7, pp. 25-33 (1992).
2. Chen, L.-H., Liao, H.-Y., Wang, J.-Y., Fan, K.-C., Hsieh, C.-C.: An Interpretation System for Cadastral Maps. *Proc. of 13th Internat. Conf. on Pattern Recognition*, IEEE Press, Los Alamitos, California, pp. 711-715 (1996).
3. Delalandre, M., Trupin, E., Labiche, J., Ogier, J.M.: Graphical Knowledge Management in Graphics Recognition Systems. In: *Graph-Based Representations in Pattern Recognition*, Lecture Notes in Computer Science (LNCS) vol. 3434, pp 35-44 (2005).
4. Ebi, N.B.: Image Interpretation of Topographic Maps on a Medium Scale Via Frame-based modelling. *International Conference on Image Processing*, IEEE Press, California, Vol. I., pp. 250-253 (1995).
5. *Graph-based Representations in Pattern Recognition*. Series of conference proceedings, *Lecture Notes in Computer Science*, Springer, last three volumes: No. 2726 (2003), No. 3434 (2005), No. 4538 (2007).
6. *Graphics Recognition (series)*. Selected papers of GREC workshops, *Lecture Notes in Computer Science*, Springer, last three volumes: No. 3088 (2004), No. 3926 (2006), No. 5046 (2008).
7. Hartog, J., Kate, T., Gerbrands, J.: Knowledge-Based Segmentation for Automatic Map Interpretation. In: *Graphics Recognition*, *Lecture Notes in Comp. Sci.* 1072, Springer, pp. 159-178 (1996).
8. Hoel, E., Menon, S., Morehouse, S.: Building a robust relational Implementation of Topology. *Lecture Notes in Computer Science*, Springer, No. 2750, pp. 508-524, (2003).

9. Katona, E., Hudra, Gy.: An Interpretation System for Cadastral Maps. Proceedings of 10th International Conference on Image Analysis and Processing (ICIAP 99), IEEE Press, pp. 792-797 (1999).
10. Katona, E.: Automatic map interpretation. Ph.D. Thesis (in Hungarian), University of Szeged (2001).
11. Lladós, J., Sanchez, G., Marti, E.: A String-Based Method to Recognize Symbols and Structural Textures in Architectural Plans. In: Graphics Recognition, Lecture Notes in Comp. Sci. 1389, Springer, pp. 91-103 (1998).
12. Messner, B.T., Bunke, H.: Automatic Learning and Recognition of Graphical Symbols in Engineering Drawings. In: Graphics Recognition, Lecture Notes in Comp. Sci. Vol. 1072, Springer, pp. 123-134 (1996).
13. Niemann, H., Sagerer, G.F., Schröder, S., Kummert, F.: ERNEST: A Semantic Network System for Pattern Understanding. IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 12, No. 9, pp. 883-905 (1990).
14. Open Geospatial Consortium: Simple Features Specification for SQL – Version 1.1. <http://www.opengeospatial.org>.
15. Qureshi, R.J., Ramel, J.Y., Cardot, H.: Graph Based Shapes Representation and Recognition. In: Graph-Based Representations in Pattern Recognition, Lecture Notes in Computer Science (LNCS) vol. 4538, pp 49-60 (2007).
16. Samet, H.: Design and Analysis of Spatial Data Structures. Addison Wesley (1989).
17. Schawemaker, J.G.M., Reinders, M.J.T.: Information Fusion for Conflict Resolution in Map Interpretation. In: Graphics Recognition, Lecture Notes in Comp. Sci. 1389, Springer, pp. 231-242 (1998).
18. Suzuki, S., Yamada, T.: MARIS: Map Recognition Input System. Pattern Recognition, Vol. 23, No. 8, pp. 919-933 (1990).
19. Vaxiviere, P., Tombre, K.: Celestin: CAD Conversion of Mechanical Drawings. Computer, Vol. 25, No. 7, pp. 46-54 (1992).
20. Wenyin, L., Dori, D.: From Raster to Vectors: Extracting Visual Information from Line Drawings. Pattern Analysis and Applications, Springer, 1999/2, pp. 10-21 (1999).