A knowledge based object-oriented approach to cartographic generalization

Marc P Armstrong
University of Iowa

David A Bennett
University of Iowa
A KNOWLEDGE BASED OBJECT-ORIENTED APPROACH TO CARTOGRAPHIC GENERALIZATION

Marc P. Armstrong & David A. Bennett
316 Jessup Hall
The University of Iowa
Iowa City, IA 52242

1.0 INTRODUCTION

Detailed digital cartographic information is now widely available from public and private sector sources. Digital cartographic information also is often encoded from detailed, large scale analog source documents such as topographic quadrangles. In each case, these sources define an upper bound on the amount of geometrical detail that is present in a GIS database. It is unlikely, however, that all GIS applications will require data at the same level of generalization, and even for a single application such as map display, requirements for geometrical detail change dramatically with scale. Thus, GIS users may encounter situations in which the detail contained in a database is inappropriate for their application. Knowledge based approaches show great promise for guiding the selection and application of generalization methods to address this cartographic problem (Buttenfield, 1990; Mark, 1990; McMaster, 1990). The purpose of this paper is to describe a strategy for organizing knowledge required to support generalization, and a process for controlling the planning of generalization operations. Object-oriented programming techniques are used to accomplish these goals.

1.1 Cartographic Generalization Problems

Several obstacles to the implementation of fully automated generalization must be overcome.

• Determining an appropriate level of abstraction: The amount of change required to accomplish a generalization task depends on the objective and scale of the desired product. Brassel and Weibel (1988), for example, draw a distinction between statistical and cartographic generalization objectives. Statistical generalization is intended to support spatial analyses, whereas cartographic generalization is performed for visual communication. In each case, the required operations and their parameters may be quite different. For example, the data required for analyses may simply consist of nodes connected to form links in an abstracted network used by optimization algorithms. For map displays, however, a more geometrically
detailed network may be required to enable decision-makers to orient themselves.

- Generating an appropriate level of abstraction: Automated generalization is difficult to perform. Although a set of generalization operations is available (e.g. McMaster, 1987; Brassel and Weibel, 1988), the parameters for a given application are unknown, and may vary over a single map.

- Managing interaction among map elements: For each generalization objective, the interaction among generalization operations, and the specification of a logical sequence for implementing them in a given context may change (Brassel and Weibel, 1988). McMaster and Monmonier (1989), for example, show how the required set of raster generalization operators changes with the scale of a desired map product. Generalization operations also can cause cascading effects. For example, if a rule is invoked to displace a feature, the displacement may affect the size and location of typography associated with it, which can cause other displacements to occur.

1.2 Object-Oriented Programming

Object-oriented programming provides a means for describing spatial objects, and for representing and applying knowledge that is required to support generalization (Mark, 1990). An object consists of instance variables which describe its current state, and methods which encode logic to formalize its behavior (Blair et al., 1989; Kim, 1990; Wegner, 1987). A class is characterized by a specific set of instance variables and methods, and acts as a template for related objects. Classes of objects are organized hierarchically, and class characteristics are inherited by members of subclasses. By using the encapsulated description of an object (instance variables and methods), knowledge can be represented and applied to generalization problems.

In an object-oriented environment a map can be modeled as an abstract object defined by design objectives and subclasses of cartographic objects. When a display is required, cartographic objects are sent messages to display themselves using their encapsulated knowledge. Therefore, each object can use those algorithms and parameters that are most appropriate for the stated objectives of the map and the specific feature type. If map elements belong to a subclass of the map class, design objectives are inherited and, hence, state variables of cartographic objects need not be specified redundantly. The properties of inheritance and polymorphism (a single message causing multiple behaviors) combine to provide a flexible and efficient programming environment. The system becomes more complicated, however, when a single object can inherit multiple behaviors. Some object-
oriented languages, such as C++, allow a class to inherit characteristics from multiple superclasses. In this situation, controlling logic determines the ancestor from which a common method (e.g. display) is inherited. Other languages, such as Smalltalk/V (Digitalk, 1988), do not support multiple inheritance and therefore, multiple behaviors exhibited by a single object must be simulated either by object sharing or object coercion. As an example consider a case when a chain represents a portion of both a river and a boundary between two regions. When object sharing is used, the same chain serves as the boundary and river, whereas when an object is coerced, it is converted into an instance of another class.

2.0 GENERALIZATION KNOWLEDGE BASE

Past approaches to automated generalization are unsatisfactory because although they to specify a mechanism for generalization, they fail to provide its parameterization for a given application and a strategy for the invocation of generalization operators. One way to deal with this problem is to encode knowledge about generalization processes as part of the computer system. Knowledge can be decomposed into a set of categories to facilitate its representation, management, and use in generalization (Armstrong, 1990). Geometrical knowledge describes the characteristics of features at an arbitrarily large scale; it includes a conventional set of feature descriptions and other knowledge (e.g. feature density) that can be applied to generalization. Structural knowledge provides details about the generating processes that gave rise to each feature, and brings geographical expertise that resides with the cartographer into the automated generalization process. Procedural knowledge is used to select appropriate operators, and to guide the application of algorithms that perform generalization.

These categories of knowledge must be used together to accomplish generalization goals. Note, however, that the rules used to decide whether an object is displayed are complex and may be mediated by the density of features in a region of a map. This introduces considerable complexity into the process, and may require a conditional generalization plan, and then a subsequent resolution of conflict. Because of this complexity, interaction among facts and rules must be supported by a higher level of knowledge organization and use. This higher level knowledge enables planning about generalization strategies to take place and is required to guide the application of rules, operators and their parameters. The application of such knowledge requires the development of a metaplanner (Armstrong, et al., in press).
3.0 THE METAPLANNER

The metaplanner controls generalization processes by eliciting objectives and then creating a generalization scenario in map form that is evaluated by the user. Because instance variables and methods are encapsulated with objects, portions of the metaplanner are distributed among the classes of objects in the database. Although distributing metaplanner functions adds complexity to its design, inheritance enables methods and knowledge to be passed among objects of similar type or dimension.

The metaplanner must have access to a set of goals for it to construct a scenario which will be evaluated by the cartographic decision-maker. Specific objectives of a scenario include:

1) **Scale**: Is there a requirement that the map be produced at a specific scale, or is scale a parameter that can be adjusted to accommodate other factors? Is there a range of acceptability?
2) **Purpose**: What is the purpose of the map? Is it a thematic map which does not require high amounts of geometrical accuracy, or is it designed for a purpose that requires high degrees of accuracy (e.g. navigation)?
3) **Inclusion**: Is the user interested in specifying a list of items that must, should, might, or must not appear?
4) **Weighting**: Is there a priority of items in each of the categories?
5) **Interaction**: How do map elements interact?

The metaplanner establishes generalization goals through a dialog with the decision-maker. At this time, conflicts among design objectives must be resolved by the decision-maker.

High level metaplanning activities consist of strategies for treating entire classes of objects (e.g. 0, 1, and 2 cells). An example of a high level metaplanning activity is line generalization in which methods such as n-th point or Douglas-Peucker are encapsulated. At a lower level, cartographic objects not only inherit these general strategies, but also encapsulate strategies for generalization that are germane to specific problem instances or subclasses of objects. For example, a highway object (a subclass of 1 cells) may have a parallel line generation method to provide a cased representation. The metaplanner determines when a cased representation is appropriate and the displacement of the parallel lines.

4.0 SYSTEM IMPLEMENTATION

A prototype object-oriented cartographic system has been developed in Smalltalk/V (Digitalk, 1988). Although this system is intended only to explore the application of the object-oriented paradigm to generalization its evolution into a fully functional system is facilitated by two characteristics that typify the object-oriented approach: extendibility and continuity (Meyer, 1988). The
concepts described here are illustrated by considering the process of simplification among digital representations at different scales. Simplification refers to the process of eliminating unneeded detail from a map, and geometrical, structural, and procedural knowledge can be applied to assist in the creation of a satisfactory display. Two types of simplification are considered: line simplification, and feature elimination. Line simplification is a well-studied form of cartographic generalization (e.g. McMaster, 1987) in which the number of points in a line is reduced with the objective of retaining its large scale form. Feature elimination increases map legibility by reducing clutter and removing extraneous information. Procedural knowledge is used to select appropriate generalization operators for a given scale and design objective.

The first step in implementing an object-oriented system is to determine the structure and behavior of each object to be represented in the class hierarchy. As suggested by Meyer (1988), it is logical to model objects after features in the environment under study, in this case existing map manuscripts. This is a bottom up approach which creates, for example, a graphic distinction between rivers and roads, political boundaries and forests. Each of these basic cartographic elements are mapped to a unique object within the object-oriented system. The structure and, to a lesser extent, the cartographic behavior of these elements can be abstracted into three classes of spatial primitives (0-Cell, 1-Cell, and 2-Cell). In this manner behavior common to multiple objects is gathered into new more abstract classes, thus forming a hierarchical system of cartographic objects (Figure 1). This process is referred to as set based abstraction.

Spatial primitives standardize the geometrical knowledge used to describe cartographic elements but are too abstract to provide the structural knowledge needed to completely describe cartographic behavior. Fundamental differences in the

Figure 1. Prototype of a class hierarchy for an object-oriented cartographic system.
behavior of objects constructed by humans versus natural objects of the same spatial dimension, however, may exist (e.g. the sinuosity of a river compared to that of a road). Although the necessary structural knowledge is not yet formalized, natural and anthropogenic subclasses have been added under the 1 and 2 cell spatial primitives to support future research. The MapMaker class is added at the top of this class hierarchy to store map objectives and to coordinate the creation of cartographic products. For the purpose of this paper only one lineage in this hierarchy, the StreamReach ancestry, is discussed.

4.1 MapMaker Class

The topmost cartographic object (MapMaker) stores the design objective, a set of displayable cartographic objects, and the graphic window needed to display a map product (Table 1). The setObjective method is responsible for high level metaplanning functions which establish map objectives through a dialog with the user. The mapObjects variable is an instance of the standard Smalltalk class Set. This allows cartographic objects of various classes to be aggregated into a common data structure while insuring that no map element is duplicated. To draw a map each element in this set of cartographic objects is asked to display itself:

\[ \text{drawMap}\]

"create a map of containing those elements in the set mapObjects which are appropriate given the stated design objectives"

mapObjects do:[:aMapObject/ aMapObject display].

The polymorphic behavior of display allows each object to be mapped according to its unique state and behavior.

Table 1. MapMaker Class

<table>
<thead>
<tr>
<th>Class Variables:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective- Set of strings.</td>
</tr>
<tr>
<td>TheWorld- TopPane.</td>
</tr>
<tr>
<td>TheMap- GraphPane.</td>
</tr>
<tr>
<td>MapScale- Integer.</td>
</tr>
</tbody>
</table>

Instance Variable:

mapObjects- Set of cartographic objects.

Methods:

addToMap- add a cartographic object to the map.

initialize- set initial configuration.

initializeTheMap- initialize the map form.

setObjective- establish design objectives.

drawMap- display all mapObjects.
4.1.1 0-Cell Spatial Primitives. The Smalltalk class Point is used to model 0-cell spatial primitives. However, to make this class suitable for an automated cartographic environment two new methods were required. First, a method which returns the distance of a point from a line defined by two other points was added to support the Douglas-Peucker line generalization algorithm (Douglas and Peucker, 1973). Second, a display method was added to represent a point as a solid dot of specified dimension (using the solidEllipse method from the Smalltalk class Pen).

4.1.2 1-Cell Spatial Primitives. The 1-cell class acts as a template for all linear objects. At present, 1-Cell class objects (Table 2) are not assigned metaplanning responsibility and, thus, act to provide a common data structure and a set of tools which can be used by all subclasses (e.g. streams, roads). A chain that fills multiple cartographic roles (a stream and a boundary) is shared by multiple objects. This strategy helps maintain data integrity by implementing a topological data structure.

Table 2. 1-Cell Class

<table>
<thead>
<tr>
<th>Inherits From:</th>
<th>MapMaker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inherited By:</td>
<td>Anthropogenic, Natural</td>
</tr>
</tbody>
</table>

Instance Variables:
- linkedFeatures - Set of shared 1-cell objects.
- theChain - OrderedCollection of Points.
- leftPolygon,
- rightPolygon - Shared 2-Cell objects.
- length,
- width - Float.

Methods:
- addAPtX:y- add point (x,y) to the receiver chain.
- displayByNthPoint:N- display receiver using the Nth point algorithm.
- displayByDouglasPeucker - display receiver chain after it is generalized using the Douglas-Peucker algorithm.
- generalizeByDouglasPeucker:aChain withEps:eps - return a copy of aChain which has been generalized using the Douglas-Peucker algorithm (epsilon=eps).

4.1.2.1 StreamReach Class. The StreamReach class augments the definition of the 1-cell class by adding the variables stream_Order and discharge (Table 3). These instance variables along with encapsulated and inherited knowledge provide the information needed by the metaplanner for the selection and parameterization of the generalization operators. For example,
scale and design objectives can be used to establish rules for line simplification:

\[
(\text{Map\_Scale} > x \text{ and: } [\text{Map\_Scale} < y \text{ and: } [\text{Objective includes:'general reference']})
\]

ifTrue: [self generalizeByDouglasPeucker :0.1]

In this example Smalltalk code fragment, the Douglas-Peucker algorithm would be selected with an epsilon (0.1) to generalize a chain.

**Table 3. StreamReach Class**

Inherits From: Natural

Instance Variables:
- discharge- Float.
- stream\_Order- Integer.

Methods:
- **display** - selects and parameterizes generalization operators.

Two criteria are used to provide guidance about feature retention: geometry and attributes. Monmonier (1986) has suggested that geometry be included in a digital database so that only streams with a certain stream order would be shown at a given scale. This can be accomplished by feature tagging, or through the use of a transfer function. Features in a tagging scheme are eligible for inclusion in a display only within a range of scales. If, for example, the target scale of the map has been determined:

\[
\text{stream\_order} < 2
\]

ifTrue: ["do not plot"]

In this case a precondition is tested for compliance (stream order less than 2?) and an action is taken if it evaluates to true (do not plot).

Additional structural information also may be important in determining whether features should be included on a map. Gardiner (1982) suggested that physical characteristics such as channel width and discharge can be used to identify streams to be plotted. In this way it is possible to capture some of the rich knowledge that an expert cartographer would bring to the generalization process.

\[
(\text{width} < 20 \text{ and: } [\text{discharge} < 200 \text{ and: } [\text{Map\_Scale} < 125000 ]])
\]

ifTrue: ["do not plot"]

In this example, facts about each object can be stored in the knowledge base as geometrical knowledge. Structural knowledge is applied to these facts to determine suitable plotting strategies.
4.1.3 2-Cell Spatial Primitives. The border of polygonal features are modeled as a collection of previously instantiated (shared) linear objects. To complete the definition of a 2-Cell object each polygon also possesses an identifier, an area, and a centroid (Table 4). Two cell spatial primitives can be displayed as a dot, a polygon or not at all. When a 2-cell object is displayed as a polygon the possibility exist that a chain in the collection of edges will be shared by multiple 1-Cell objects (e.g. a stream and a jurisdictional boundary). The display behavior of such an object is controlled by the order which each object class is asked to display itself. Which display strategy is invoked depends on the knowledge encapsulated with the cartographic element, the scale, and the design objectives.

Table 4. 2-Cell Class

Inherits from: MapMaker
Inherited by: Natural, Anthropogenic

Instance Variables:

theBorder- IndexedCollection of 1-Cell objects.
name- String.
area- Float.
centroid- Point.

Methods:

addAnEdge- add a linear object to theBorder.
calculateCentroid.
calculateArea.

CONCLUSION

Automated cartographic generalization can be improved by capturing the knowledge of expert cartographers and applying it in the digital domain. Object-oriented design provides a paradigm which allows us to encapsulate knowledge within spatial databases. This integrated abstract data type generates a tight coupling among feature types, their cartographic representation, a set of generalization operators and knowledge about how each operator should be applied in the context of map scenarios that are developed with user input. The approach extends traditional automated generalization processes by overcoming problems associated with the selection of appropriate operators and parameters. The knowledge base, however, is not fully supported by theory, and relies instead on empirical observations. Further theoretical development of generalization principles will enable a more robust knowledge base to be developed.
REFERENCES


