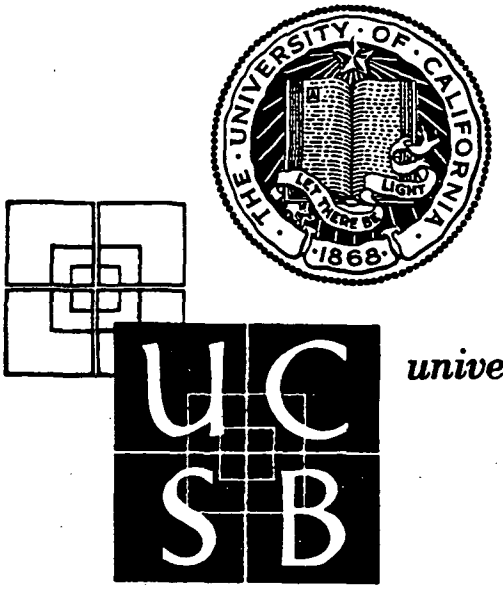
D2-43

N86-32865

KBGIS-II: A Knowledge-Based Geographic Information System

Terence Smith, Donna Peuquet, Sudhakar Menon
and Pankaj Agarwal



university of california • santa barbara

Department of Computer Science

TRCS86-13

KBGIS-II: A Knowledge-Based Geographic Information System

Terence Smith
Donna Peuquet
Sudhakar Menon
Pankaj Agarwal

College of Engineering

KBGIS-II : A KNOWLEDGE-BASED GEOGRAPHIC INFORMATION SYSTEM

Terence Smith
Donna Peuquet
Sudhakar Menon
Pankaj Agarwal

University of California, Santa Barbara

ABSTRACT

In this paper we describe the architecture and working of a recently implemented knowledge-based GIS (KBGIS-II) that was designed to satisfy several general criteria for GIS. The system has four major functions that include query-answering, learning and editing. The main query finds constrained locations for spatial objects that are describable in a predicate-calculus based spatial object language. The main search procedures incude a family of constraint-satisfaction procedures that use a spatial object knowledge base to search efficiently for complex spatial objects in large, multilayered spatial data bases.These data bases are represented in quadtree form. The search strategy is designed to reduce the computational cost of search in the average case. The learning capabilities of the system include the addition of new locations of complex spatial objects to the knowledge base as queries are answered, and the ability to learn inductively definitions of new spatial objects from examples. The new definitions are added to the knowledge base by the system. The system is currently performing all its designated tasks successfully, although currently implemented on inadequate hardware. Future reports will detail the performance characteristics of the system, and various new extensions are planned in order to enhance the power of KBGIS-II.

May 19, 1986

KBGIS-II : A KNOWLEDGE-BASED GEOGRAPHIC INFORMATION SYSTEM

Terence Smith
Donna Peuquet
Sudhakar Menon
Pankaj Agarwal

University of California, Santa Barbara

1. INTRODUCTION

In its simplest form, a geographical information system (GIS) may be viewed as a database system in which most of the data is spatially indexed, and upon which a set of procedures operate in order to answer queries about spatial entities represented in the database. On the basis of previous research concerning the design and implementation of GIS, one may infer several requirements that a GIS should satisfy, as well as several principles of design and implementation that permit the satisfaction of such requirements. In this essay, we examine both the requirements and the associated principles, first in general terms and then in terms of a knowledge-based GIS (KBGIS-II) that has been recently implemented.

1.1. Requirements of GIS

Previous research (see, for example, Marble[14], Caulkins[3] and Peuquet[17]) suggests that the following general requirements should be satisfied in the design and implementation of most GIS:

a) an ability to handle large, multilayered, heterogeneous databases of spatially indexed data

b) an ability to query such databases about the existence, location and properties of a wide range of spatial objects

This work was partially supported by USGS under the grant USDI-US9S-21167-5, NASA under NCC2-359 and NSF under NSFSES84-00799

c) an efficiency in handling such queries that permits the system to be interactive

d) a flexibility in configuring the system that is sufficient to permit the system to be easily tailored to accomadate a variety of specific applications and users.

The preceding requirements imply that any GIS satisfying them to a significant degree will be a large and complex software system, designed to run on a hardware system with both extensive memory and fast processing capabilities. Hence the design, construction and testing of the software will be a large and complex task requiring the systemmatic application of techniques developed in computer science.

1.2. Principles for satisfying the requirements

There are several general principles that may be applied in order to facilitate the design and implementation of a GIS satisfying the four requirements listed above. A first principle, relating to all four of the requirements, involves the systematic application of techniques and approaches developed in a variety of subfields of computer science (CS). To date, few GIS have been constructed on the basis of such systematic knowledge. Five subfields of CS appearing to have particular relevance for GIS include:

a) Software engineering, which provides a set of techniques to aid in the design, implementation and testing of large software systems. Only recently have GIS researchers (eg Aronson[1], Caulkins[3], and Marble[14]), described the applicability of software engineering techniques to the construction of GIS.

b) Database theory, which provides a selection of data models (see Peuquet[17]), data structures and database management techniques that may be used in satisfying the first three requirements listed above.

c) The study of algorithms and complexity is applicable to GIS in its provision of a theoretical basis for algorithms that will search large spatial databases for complex spatial objects in an efficient manner. In particular, the emerging subfield of computational geometry (see Preparata and Shamos[18]) promises much in the way of efficient spatial algorithms.

d) Artificial intelligence studies computational techniques for solving problems which are either computationally intractable or for which there are no well-understood algorithms. The complexity of spatial objects and the size of the spatial databases suggests the applicability of AI techniques in designing data-structures and procedures for answering queries.

e) Computer graphics and natural language processing are subfields of CS that provide techniques for constructing efficient and appropriate interfaces to GIS.

A second principle, relating to the first three requirements listed above, involves the integration of approaches and procedures developed in a variety of disciplines that are related to GIS. These disciplines include computer vision, image understanding and digital cartography (see, for example, Ballard and Brown[2]). Two reasons for this integration are:

a) these disciplines all study the same basic problem of recognizing and reasoning about spatial objects implicitly encoded in spatially indexed data sets. Since their evolution has been somewhat independent, GIS research would benefit from the integration of approaches and procedures developed in these other disciplines.

b) There has been a recent and growing realization that it is often a practical necessity to merge image data sets, such as LANDSAT scenes, with the more traditional datasets of GIS, such as digitized maps and vectorized representations of map features (see Jackson[11]). Computer vision and image understanding have developed techniques that will allow the integration of such capabilities into GIS.

A third principle, relating to the third requirement, involves the application of procedures that reduce the search effort involved in answering queries, particularly by avoiding simple, exhaustive search strategies. As we note below, responding to queries about complex spatial objects in a large database is an inherently difficult computational task. One approach to reducing search effort involves the application of various knowledge-based search techniques developed in AI research that employ the empirical and theoretical knowledge developed in several substantive fields of study, such as forestry, geography, geology and geophysics.

A final principle, relating to the fourth requirement, is to construct GIS in such a way that they may be easily tailored to specific applications and/or users by the users themselves. In particular, one may provide editors that allow users to augment and modify the system's data and knowledge structures. One may also provide "learning" procedures that automatically augment the system's data and knowledge structures as queries are processed.

1.3. Structure of the Essay

In the main body of this essay, we discuss these requirements and principles in terms of a knowledge-based GIS (KBGIS-II) which has just been implemented. We first provide an overview of the system, including the main system functions and the system architecture. We then describe the language in which we represent spatial objects. In the sections following, we provide descriptions of the main components of the system, including the user interface, the spatial object knowledge base, the system editors, the high-level search procedures, the constaint-satisfaction search procedure, the low-level search procedures and the learning procedures. We conclude with a summary of the system, and its relationship to the four requirements and the associated principles.

2. OVERVIEW OF KBGIS

In this section, we provide an overview of the main functions and architecture of KBGIS-II, together with a summary of the manner in which the four requirements discussed above are met in the system.

2.1. System Functions

KBGIS-II is able to perform four main functions over which the user has control:

a) In query mode, the system answers queries concerning spatial objects that are represented, usually in implicit form, in the spatial database. At present, there are two main forms of query, which may be viewed as functional inverses. The first query takes the general form:

(FIND locations <# of cases> <spatial object> <spatial window>) (1)

and is satisfied when the system finds sets of spatial locations at which the spatial object description is satisfied for the required number of examples in a specified spatial window. The spatial object is specified in terms of the spatial object language (SOL) defined below. The inverse query takes the general form:

(FIND objects <spatial window> <object class>) (2).

which is satisfied when the system finds all spatial objects that belong to a given class of spatial objects and that exist in a specified spatial window.

A very large class of spatial data-base queries may be expressed in terms of queries (1) and (2), which include queries relating to decision-making tasks in which one seeks sets of locations that satisfy various constraints or optimality conditions.The first query, for example, may be used to find solutions to the travelling salesman problem. Furthermore, it is easy to satisfy an even broader set of queries, such as requests for statistical summaries of the spatial objects in given areas, by further processing the outputs resulting from queries (1) and (2).

b) In learn mode, the system modifies and augments its knowledge base. In one form of learning, which occurs by default in query mode, the system augments its knowledge base with the locations of a selected subset of newly discovered spatial objects. In a second form of learning, that currently must be invoked by the user, the system learns inductively how to define new spatial objects. The definitions of these new objects, and related information, are then added to to the system's knowledge base.

c) In edit mode the user is able to modify and augment the SOL and associated procedures as well as modifying the system's knowledge base.

d) In trace mode the user is able to follow the processing steps being executed by the system. Trace mode may be invoked in query, learn or edit modes.

2.2. Architecture of the System

The basic architecture of the system is illustrated in Figure 1. The user interface is a general module that controls the I/O behaviour of the system, including the parsing of user queries. Each of the four sets of procedures corresponds to one of the four main functions of the system. The function knowledge base contains knowledge about the functions that define the SOL, and is modifiable by the user. The spatial object knowledge base contains knowledge about spatial objects (such as their definitions and various heuristics), while the location tree data base contains the basic spatial data layers.

2.3. KBGIS-II and GIS Requirements

The requirement that the system handle very large, multilayered databases must be met partly in terms of the software system and partly in terms of the hardware on which the software runs. The requirement that the system be able to respond to queries about complex spatial objects is met in terms of the SOL, the search procedures adopted and the knowledge and data base structures employed. The requirement concerning search efficiency is also met in terms of the search procedures and data structures chosen, while the requirement of system flexibility is satisfied in terms of the editors available to the user. The requirement that the system handle large, multilayered databases must be met partly in terms of the software system and partly in terms of the hardware on which the software runs. The software design entailed by the requirements described above has of necessity made the current hardware (VAX 11-750) sub-optimal for the task, and the size of the databases that can be handled at interactive speeds is thus limited.

3. THE SPATIAL OBJECT LANGUAGE

Before describing the components of the system represented in Figure 1, we provide a description of the spatial object language (SOL) that is used to represent objects in KBGIS-II. The choice of SOL is important for several reasons, including:

a) The SOL defines the class of spatial objects about which the system may learn and about which the system may be queried.

b) The choice of SOL has practical implications for the ease with which various computational tasks, such as search, may be carried out.

c) The SOL is of value in revealing the computational complexity of the problem of finding spatial objects in the systems data bases.

In this section, we describe the SOL in terms of its ability to represent spatial objects.

An important feature of the SOL described below is the flexibility that it offers the user. As in similar predicate calculus-based languages (see, for example, Charniak and McDermott[5]) the syntax is relatively simple and inference mechanisms are well-known. The user, however, has the option of defining a large numbers of predicates, functions, variables and constants in order to provide the language with an expressive power that is appropriate for a given spatial domain.

3.1. The SOL defined

A spatial object is defined as a set of spatial locations together with a set of properties characterizing those locations. In its most basic form, we define a location to be a set composed of some collection of the smallest spatial units, or "pixels", that partition the area represented in the database. A location is not necessarily a connected set of pixels. One may then extend this definition of a location to include sets of locations.

We employ three classes of properties in defining the SOL:

a) Pixel properties, or PPROPs, are properties that characterize individual pixels in the database. Each layer in the spatial database has at least one associated PPROP. Examples of PPROPs are Landuse, Geology and Elevation. It is evident that the type of landuse, lithology or elevation are all properties that may be used to characterize either a single pixel or each pixel in a collection of pixels

b) Pixel-group properties, or GPROPs, are properties that characterize the collection of pixels comprising some location, but do not characterize each single pixel in the collection. Examples of GPROPs are Size, Shape and Orientation.

c) Relational properties, or RPROPs, are properties that describe the relationship between two locations or between the properties of two locations. Examples of RPROPs include Distance, Direction and Containment.

In the SOL, a spatial object is described as a conjunction of members of the three classes of properties that are applicable in characterizing a given set of spatial locations. We represent these properties in terms of predicates that may be interpreted in terms of relationships between one spatial location and a set of property values, between two spatial locations and a set of property values or between the property values of two spatial locations. PPROP and GPROP properties may be represented in the form:

EQUAL ((U-FUNCTION LOC1) VAL)

while RPROP properties may be represented either in the form:

EQUAL ((B-FUNCTION LOC1 LOC2) VAL)

or in the form

EQUAL ((B-FUNCTION <function of LOC1> <function of LOC2>) VAL)

In these definitions, LOCi is a constant or variable representing a location; VAL is a constant or variable representing the value of some property; U-FUNCTION is a unary function of one location; B-FUNCTION is a binary function of two locations; and EQUAL is a predicate that indicates the truth or falsity of the statement.

We now provide examples of the three classes of predicates:

a) To describe a location whose landuse is agriculture, we use the PPROP predicate

EQUAL ((LAND LOC1) AGRICULTURE)

This predicate is satisfied when the variable LOC1 is bound to a location (ie a set of spatial

indices) for which it is true that the value of the landuse property is AGRICULTURE for each spatial index in the location. It is possible to verify the truth value of a PPROP predicate based on information stored in the appropriate layer of the spatial database.

b) To describe a location whose area is between 50 and 60 resolution units we use the GPROP predicate

EQUAL ((AREA LOC1) (50 60))

This predicate is satisfied if the variable LOC1 is bound to a location having an area of between 50 and 60 pixels. The truth value of a GPROP predicate may be verified using computed or stored information. The system has a function for each GPROP, that computes the value of the corresponding property.

c) To describe an object consisting of two locations that are separated by a distance of 10 to 20 resolution units we use the RPROP predicate

EQUAL ((DISTANCE LOC1 LOC2) (10 20))

which is true when the locations bound to LOC1 and LOC2 are separated by 10 to 20 units. The system has a function that computes the value of the property corresponding to each RPROP.

The language also permits relational comparisons to be made between the properties of two groups of spatial indices using the arithmetic comparison operations EQ, GT, LT, GE, LE corresponding to =, >, <, >= and <= respectively. To specify, for example, that the area of one component of an object is greater that the area of another component, we may write:

EQUAL ((GT (AREA LOC1) (AREA LOC2)) TRUE)

Any of the predicates described above may be combined using the logical connectives /\ (AND) and \/ (OR) . Logical negation (-) may be combined with any PPROP or GPROP predicate by using the NOT-EQUAL predicate in place of the EQUAL predicate in the above expressions. As a simple example, we may choose to model a city as a commercial core (LOC1)

surrounded by a residential annulus (LOC2), in terms of the SOL representation

EQUAL ((LAND LOC1) COMMERCIAL)

/\ EQUAL ((AREA LOC1) (30 40))

/\ EQUAL ((LAND LOC2) RESIDENTIAL)

/\ EQUAL ((AREA LOC2) (50 60))

/\ EQUAL ((CONTAINS LOC2 LOC1) TRUE)

It is to be emphasised that the set of functions and arguments with which a spatial object may be represented in the system is definable by the user by way of the various editors.

3.2. The Spatial Object Hierarchy

We now define a special GPROP called "TYPE" that allows us to define high-level spatial objects that are themselves defined in terms of the basic P-, G- and R-PROPs. Hence we may partially order spatial objects, and so impose a hierarchical structure on them. In its simplest application, TYPE ascribes a name to a spatial object that is defined as a conjunction of PPROPS, GPROPS and RPROPS with specified values. An example of a high-level spatial object is:

((TYPE X) GEOL-OBJ1)

$\Leftrightarrow$

/\ ((LAND X1) FOREST)

/\ ((AREA X1) LARGE)

/\ ((SHAPE X1) CIRCULAR)

/\ ((GEOL X2) 4)

/\ ((ELEV X2) (50 100))

/\ ((AREA X2) MEDIUM)

$\wedge$ ((DISTANCE X1 X2) (60 100))

$\wedge$ ((DIRECTION X1 X2) NORTH)

(the predicate EQUAL is implicit, but omitted in this statement).

The above definition states that any set of locations X1 and X2 satisfying the unary and binary constraints specified on the right hand side, constitute a location X of the high-level object named GEOL-OBJ1. The relationship between the location X and the locations X1 and X2 may be chosen in some appropriate manner. For example X may be the convex hull of X1 and X2, the union of X1 and X2, or the centroid of X1 and X2. The unary constraints on the location X1 are specified by the two GPROP functions Area and Shape, and on the location X2 by the GPROP function Area. constraints on the locations X1 and X2 are specified by the two RPROP functions Distance and Direction.

In general, high-level spatial objects may be defined in terms of other high-level objects using the TYPE property, in conjunction with other PROPs, GPROPs and RPROPS.

The use of the TYPE property in assigning a name to a high-level spatial object accomplishes two objectives:

a) it provides a convenient shorthand notation by means of which objects may be defined in terms of previously defined objects. Given for example that two objects named LAND-1 and LAND-2 have been defined, it is then possible to specify a new high-level spatial object LAND-3 as follows:

((TYPE X) LAND–3)

$\Leftrightarrow$

$\wedge$ ((TYPE X1) LAND–1)

$\wedge$ ((TYPE X2) LAND–2)

$\wedge$ ((DISTANCE X1 X2) (20 30))

b) The TYPE property allows us to store newly found locations for high-level objects in a database indexed by object name and location. The indexing by location is achieved with a discrimination net, with each high-level object having its own discrimination net. These data structures are described below.

Any high-level spatial object may thus be seen to form the root of a tree, the complete expansion of which yields leaves which are PPROPs, GPROPs and RPROPs. On this basis,we may then assign each high-level spatial object some measure of its complexity that takes into account the height of the tree that links it to the leaves, the number of component objects at each level in the tree, and the complexity of the spatial relations (RPROP predicates) at each level.

For the purposes of describing the spatial object search process (see below), it proves convenient to distinguish between high level spatial objects and primitive spatial objects. Any object that has been defined in the Spatial Object Database and hence has a name which is the value of the TYPE property, will be referred to as a high level spatial object. The term primitive spatial object will be used to refer to any connected set of pixels represented by some conjunction of PPROPS. It is easy to see that any high-level object may be ultimately defined in terms of primitive spatial objects, and an appropriate set of RPROPS and GPROPS.

4. THE USER INTERFACE

The User Interface allows the user to select from among the four main functions of the system (querying, editing, learning and tracing), and to supply the appropriate inputs and outputs. At present most user inputs into the system are by way of a key board, while outputs from querying the system are displayed on a graphics device.

4.1. Querying

In query mode the user may select one of the two fundamental queries (1), (2). Queries of both types may be entered either interactively or from a file.

4.2. Editing

In edit mode, the user may modify either the SOL and associated procedures, using the Function Editor, or the system's knowledge base, using the Object Editor.

4.3. Learning

In learn mode, the user may cause the system to learn a definition of a new spatial object from given examples. Either the system searches for and generates these examples, or the user provides the examples.

5. THE KNOWLEDGE AND DATA BASES

5.1. The Spatial Object Knowledge Base

The Spatial Object Knowledge Base stores both the definitions of, and useful information about, all objects known to the system. This knowledge base is implemented in terms of a slot and filler data structure (Nilsson [16]) and a discrimination net data structure (Charniak et. al.[4]). Information concerning object definitions, search heuristics, object classification, and object complexity, as well as low level search procedures that may be directly invoked in searching for spatial objects, is stored in the knowledge base. Information concerning known locations of spatial objects that have been previously found are stored in the discrimination net database.

The slot names and information stored in the slot and filler data structures are shown in Figure 2. The information in each slot can be augmented, modified or deleted by means of the spatial object knowledge base editor. The information stored in this database may also be modified in the inductive learning mode of the system.

The discrimination net database is used to store locations of known examples of spatial objects that are generated by the system during the course of answering user queries. Each object has its own discrimination net. The keys for the discrimination net are derivable given the name of a spatial object and the desired location tree address in the database.

5.2. The Function Knowledge Base

The Function Knowledge Base stores information on functions used by the system in searching for spatial objects. Information on the functions that evaluate the GPROP and RPROP properties of spatial objects are stored in this knowledge base.

The user has the ability to add, modify and delete information from this database using a function editor. Information on the ability to propogate constraints, the computational complexity, subroutine names, symmetry, range and learning related information are stored for each GPROP and RPROP function. The slot names and information stored in the knowledge base are shown in Figure 3. The system utilizes this information to control search for spatial objects and to generate information in learn mode.

5.3. The Location Tree Data Base

The Location Tree Data Base stores information on the spatial distribution of both region based PPROPS and linear features existing within the area covered by the database.

5.3.1. Region Data

The raw input for the region based PPROPS from which the location tree data base is built consists of a raster image for each layer such as landuse, geology or elevation. The conceptual data model utilized for data storage is the quadtree structure. This data structure is based on a recursive partitioning of space into four quadrants, and has been discussed extensively in the literature (see, for example, Samet[19, 20, 21, 22], Hunter and Steiglitz [10] and Tanimoto and Pavlidis [24]) The location tree database extends the quadtree concept allowing for the encoding

of multiple layers of thematic information, with more than one class of information on each layer being stored at an internal node of the location tree. As discussed in the section on the spatial object language, the PPROPS represent primitive pixel properties such as landuse, geology and elevation. There is a layer in the location tree corresponding to each such PPROP in the database. Each node in the location tree is structured as a three dimensional frame One slot is allocated for each PPROP in the database. Each layer (slot) in turn is a frame which contains the following slots :

a) The VALUE slot stores the data values that occur in the area represented by the node. Each PPROP is quantized to have a maximum of fifty discrete values. At each intermediate node in the tree, a list of values occurring below the node, (together with the areal extant of each value) is stored. The data values are not averaged before storing as in the construction of the pyramid data structure described in Tanimoto[24]. The availability of the areal extant of each data value allows the dynamic computation of the color of a node. Thus a node may be classified as black, white or grey with respect to a particular data value depending on a variable percentage threshold.

b) The DISTRIBUTION slot stores information on the areal extent of each data value in the area represented by the node. The DISTRIBUTION slot may be used to store more than one statistic for describing various aspect of the distibution of data values. The information stored in the DISTRIBUTION slot is used to compute node color based on flexible criteria as described above.

During search to satisfy a query, each node visited by the search process is tagged using a search-tag. Allocation of space for these search-tag fields is a dynamic process and occurs during search. A unique search-tag field is used for each primitive object (connected region) that is part of a query. The information stored in the search-tag field is valid only during the dynamic extent of a query and may be removed and the space deallocated on completion of the search.

5.3.2. Linear Data

The raw input for the linear based data consists of binary raster images of each linear feature such as roads and streamlines. This data is converted to vector form through edge following procedures and the resulting vector representations are stored in spatially indexed form, as properties of the nodes in the higher levels of the Location Tree Data Base. Each vector representation of a linear feature consists of a series of straight line segments. These segments are stored in an array and cursors uniquely identify each breakpoint between segments. It is these cursors that are stored in the nodes of the of the Location Tree Data Base, permitting efficient retrieval of the subset of streams or other linear features within any specified block of the database.

6. EDITORS

KBGIS-II provides two editors, the Function Editor and the Spatial Object Knowledge Base Editor. The Function Editor permits the user to modify the function knowledge base and the Spatial Knowledge Base Editor permits the user to update the spatial object knowledge base. These editors are menu driven, and the user may alter the knowledge bases by selecting any of five modes.

a) In ADD mode, the Object Editor may create a new spatial object. It queries the user for the FeatureType of the new object. An object definition package is then invoked and the user is guided through the construction of the object's DefinedBy slot in terms of the SOL. Besides the definition, the editor also asks for other information such as class, heuristics, and linear and areal dimensions. In this mode, The Function Editor adds a new GPROP or RPROP. The user specifies the file name which contains the definition of the functions. If the new function propagates the constraint, the file should contain a function which can return a new search window. Besides the function definition, the editor also asks for associated parameters such as complexity, symmetry and domain.

b) In DELETE mode the Object Editor deletes spatial objects from the knowledge base. The deletion of an object is allowed by the system only if it is not currently used as a component

in the definition of any other spatial object in the knowledge base. The Function Editor deletes GPROPS and RPROPS from the function knowledge base.

c) In MODIFY mode the user may modify the contents of any slot of either a selected spatial object or a function. The system ensures that logical consistency is maintained before allowing modifications to be made.

d) In DISPLAY mode the user is allowed to browse through the knowledge base, examining selected components of selected objects or functions.

e) In HELP mode the user is provided with aid in using the editors.

f) In END mode, the user may save the changes made in the current session.

7. SPATIAL SEARCH

It is clear from the preceding discussion that procedures that search for spatial objects lie at the core of GIS in general and of KBGIS-II in particular. In this section of the essay, we briefly outline the major principles and procedures that underly the search for spatial objects in KBGIS-II. It should be recalled that search efficiency is a major requirement in most GIS.

7.1. Principles of Search

Smith and Peuquet [23] outlined five principles that underlie the search procedures in KBGIS-II. We repeat those principles here with one further addition:

a) The use of hierarchical decompositions in both data structures and in the search procedures applied to the data structures.

b) The availability of different search strategies that may be chosen as the most efficicent in a given search context.

c) The application of best first search procedures in which domain-specific knowledge is used to reduce the sets of locations that need to be searched in answering queries.

d) The use of a constraint-satisfaction approach

e) The use of recursion

f) The use of dynamic updating of the system's knowledge base in response to query satisfaction.

The application of these principles is implicitly described in the detailed descriptions of the search procedures that are provided in following the sections.

7.2. Search Procedures

For convenience, we now provide a brief overview of the search procedures, based on the six principles enunciated above, that are employed in KBGIS-II when satisfying queries of type (1). When a query is entered by a user, it is parsed and checked for syntactic correctness and the user is prompted for any modifications. The (high-level) object of the query is then transformed into a semantic network representation, in which links represent RPROP relations (or constraints) between the subobjects of the query that must be satisfied. The network is then augmented with heuristic knowledge and the subobjects at the nodes are ordered. A constraint satisfaction procedure is then applied to the nodes in the designated order. Search first occurs in the system knowledge base for specific subobjects that are known to satisfy the relational and spatial constraints. If the satisfaction of the query cannot be accomplished by this lookup procedure, the search procedure is recursively called on the subobjects of the node. The recursion terminates in procedures that search the location tree database of the system. When a query is ultimately satisfied by a search of this database and when the search is considered computationally expensive, the result is stored in the system's knowledge base for use in future search.

In the above search process constraint satisfaction procedures are used to satisfy all unary (GPROP) and binary (RPROP) constraints used in the definition of an object, and as such provide the core of our approach to spatial search. The general constraint satisfaction problem (CSP) has been studied by many researchers, including Mackworth [13, 15] and Haralick et. al.[6, 7] The problem may be stated as follows[15]

> Given a set of m variables each with an associated domain and a set of constraining relations each involving a subset of the variables, find all possible m-tuples such that each m-tuple is an instantiation of the m variables satisfying the relations.

Mackworth considers only CSP's that are discrete, finite and for which the relations are unary and binary.

The classical approach to the CSP entails the use of backtracking. The variables are instantiated in sequential order using labels selected from an ordered representation of the domain. Backtracking therefore corresponds to a depth first search of the combinatorial search space, with the truth values of intermediate predicates being tested in order to terminate unsuccessful searches as early as possible. As soon as the variables of any predicate are instantiated, the truth value of the predicate is tested. If true, then the process of testing and instantiation continues, but if false the process falls back to the variable last instantiated that has untried values in its domain and and reinstantiates it to its next value.

Although the intrinsic merit of backtracking is that substantial portions of the generate and test search space (the cartesian product of all the variable domains) are eliminated by a single failure, it may still be very inefficient. Various improvements to the procedure have been suggested, such as preprocessing the network for node, arc and path consistency (see Mackworth[13, 15]) and forward looking tree search which prunes the search space through the use of a look ahead procedure (see Haralick[8])

We discuss our approach to spatial search in more detail in the following sections, first in terms of the high level search procedures that control search, then in terms of the constraint satisfaction procedures and finally in terms of the low- level procedures that search the location tree database.

7.3. The SOL and Search Procedures

The structure of the SOL may now be viewed in terms of its relation to the search procedures. First, the use of a language that involves only unary (PPROP, GPROP) and binary (RPROP) relations allows the immediate construction of semantic network representations of the spatial objects. These representations have a natural spatial interpretation and provide a data structure upon which constraint satisfaction techniques may be naturally applied. Second, the use of the TYPE predicate in the SOL permits the natural use of recursive calls during the process of query satisfaction.

7.4. The Complexity of Spatial Object Search

As noted above, the SOL is of value in indicating the computational complexity of the search for spatial objects. By the complexity of search for a given object, we shall mean a measure of the computational time that is required to find such an object, stated as a function of some measure of the object's size. We now provide a simple and heuristic argument indicating that the search for spatial objects is in general a very difficult computational problem. We show by way of an example that it is easy to construct spatial objects that have a very simple representation in terms of the SOL defined above, and a very high order of search complexity.

We may conceive of a spatial object that is comprised of n subobjects, which are linked in such a manner as to give rise to a connected graph. We shall use the number of subobjects (n) as the measure of the size of the spatial object. The links between subobjects may be represented in terms of some RPROP. We may further assume that each of the subobjects is characterized by some GPROP that can take on two values with equal probability. If we assume that the subobjects are distributed at random in our spatial database, then the probability that any given location satisfies a GPROP constraint, (and hence constitutes an example of the corresponding subobject) is 1/2. The probability that n locations, in the configuration specified by the RPROPS, satisfy the GPROP constraints is hence $(\frac{1}{2}^{n})$. In the absence of preprocessing, and assuming that subobjects are located at random in our spatial database, it is necessary to examine each n-

tuple of locations (that lie in the configuration specified by the RPROPS) to check whether the GPROP constraints are satisfied. It follows that we will have to search $O(2^n)$ times on average before finding an object with a set of nodes having the prescribed GPROP values. Furthermore, search could take significantly longer in some cases, and it is easy to express much more complicated objects in the SOL.

Reduction in this time complexity is possibe if additional information is available to the search process. If subobjects are not distributed at random in the database then such information may be created by preprocessing and/or by making heuristic knowledge on the distribution patterns of objects available to the search process. Heuristic knowledge, in the above example, may consist of storing windows for each subobject where the probability of a location satisfying the GPROP (PPROP) constraints necessary to make it an example of the subobject are higher than for the rest of the database. Similarly a stored window for the parent spatial object will indicate a higher probability, within the window, that n-tuples of locations that lie in the spatial configurations specified by the RPROPS satisfying the GPROP constraints. Within this stored window there is thus exploitable correlation in the locations of subobject.

Despite the possible speedup in search made possible by such preprocessing, the inevitable conclusion of the preceding remarks is that the search for arbitrary spatial objects describable in terms of our SOL is a problem with a high order of computational complexity.

8. HIGH LEVEL OBJECT SEARCH

High Level Object Search is the procedure used to search for locations of any high level object and is used to satisfy a query of type (1). It is first called upon to find examples of the 'Query' object. It may be called recursively if the 'Query' object has other high-level objects as its descendents. The level of recursion permitted in the search process is unlimited.

The first step in spatial search is to reduce the size of the search window using available knowledge concerning the locations of objects. This is accomplished by accessing the object knowledge base to find other high-level objects that are contextually related to the object sought.

The system then determines if any known examples of such ancilliary objects exist within the search window. If so, sub-windows are constructed around each of the ancilliary locations, and are employed as likely areas for search. Hence a queue of windows is constructed, and the the system searches sequentially for the object in each of these windows until the required number of examples of the object are found. For any one window this task may be accomplished by the high-level object search procedure in two ways:

a) Known locations of the object in the specified window of the spatial database may be retrieved from the spatially indexed knowledge base of known examples described above. The set of known locations stored in this knowledge base is not complete, and depends on the history of previous searches. At any time, this set generally contains only a fraction of the examples of the objects that exist implicitly in the spatial database.

b) New locations of the object may be discovered through the process of search in the window. The process of searching for a new location of a high-level object with **m** sub-objects entails discovering **m** locations, one for each of its sub-objects, such that this set of locations satisfy all unary and binary constraints that define the parent object. If the query requires searching for **n** examples of the parent object, then **n** such sets of **m** locations each must be found. Searching for a new location of the parent object given a set of candidate locations for each of the **m** sub-objects is a constraint satisfaction problem. This problem consists of an allocation of locations to sub-objects from their candidate sets such that when all **m** assignments have been made, all constraints on and between sub-objects are satisfied. The next section will provide details on the design of the constraint satisfaction procedure implemented in KBGIS II.

The task of determining which candidate locations for any one of the **m** sub-objects to employ in the constraint satisfation procedure, is a recursive specification of the task of determining locations of a high-level spatial object. The recursion terminates in the task of determining the locations of a primitive spatial object. Known examples of such objects are not stored and their locations are always determined through a search of the spatial database. This search

involves the determination of a connected set of pixels satisfying a conjunction of disjunctions of PPROP predicates and is achieved through an appropriate region growing process. Details of this primitive object search procedure are presented in a later section.

High level object search may be represented in terms of the tree shown in Figure 4. We consider the task of finding new locations of the root high-level object O, shown in Figure 4, in a window. It is assumed that heuristic knowledge has already been applied to constrain the size of the window as described above. The number of sub-objects of a parent object is not bounded, and varies with the TYPE of the object, but has been taken as three in this example. This task may be addressed by using a constraint satisfaction procedure taking as input the locations of its sub-objects o1, o2 and o3, and as constraints the binary spatial relations that link o1, o2 and o3. The locations of the sub-objects o1, o2 and o3 that serve as input to the constraint satisfaction procedure may be known examples from the spatially indexed database of known examples or new locations discovered by search.

Searching for new locations of o1, for example, is a recursive application of this task with o1 as the parent object and o11, o12 and o13 as the sub-objects. The recursion terminates, for example, at o11, which is a primitive object and is searched for directly in the spatial database.

The above procedure is followed in the search for new examples of all defined high level objects except in those cases where special purpose search procedures exist. Information on these procedures is stored in the Spatial Object Knowledge Base and is available to the control process. In these cases the special search function is directly called. The examples returned are absorbed into the constraint satisfaction process if the object in question was a subobject of some parent object being searched. This is the way in which defined objects that are linear features are searched for. This ability to interface to external search routines allows the system to utilize efficient special purpose algorithms that may be applicable in the search for a user defined object. In these cases the user may provide the system with necessary knowledge concerning the special purpose function through the function editor.

9. CONSTRAINT SATISFACTION

We now consider the task of constraint satisfaction at any intermediate level in the hierarchy shown in Figure 4. For concreteness, we consider the procedure operating on the sub-objects o1, o2 and o3. These sub-objects are subject to both unary (GPROP) and binary (RPROP) constraints. The high-level object search on the parent object O converts its definition into a semantic network, as shown in Figure 4 and this network, with o1, o2 and o3 as nodes, is passed to the constraint satisfaction procedure. Each node is linked by spatial relations (constraining arcs) to its siblings, and by parent and child links to the nodes immediately above and below it in the hierarchy. The child nodes are created only if the search procedure is recursively called on any of o1, o2 or o3. The constraint satisfaction procedure is concerned only with the spatial relations and operates on the set of nodes that are siblings (i.e. o1, o2 and o3).

The above constraint satisfaction problem for spatial objects may be mapped onto the general constraint satisfaction problem described in a previous section of the paper. The variables represent the locations of the **m** sub-objects of a parent object while the domain of each variable is the set of candidate locations for the sub-object. A feature of the spatial search problem is that the knowledge possessed by the constraint satisfaction procedure concerning the variable domains (the set of candidate locations of each sub-object) may be partial. The spatial constraint satisfaction procedure may not generally assume that it is working with all the possible values of each of the **m** variables. Through exhaustive search, it is possible to determine all locations of each sub-object in the window, before beginning the backtracking search for **m** tuples of locations that satisfy the constraints necessary to form an example of the sought for parent object. This may be appropriate if one is searching for all examples of the parent object in the window, but is inappropriate if one is searching for a small number of instances of the parent object. In the latter case the cost of exahustive search for all examples of sub-objects in a large database before beginning the constraint satisfation task for the parent objects may be computationally expensive and unwarranted. The spatial search procedure in KBGIS II therefore dynamically selects a constraint satisfaction strategy based on the nature of the spatial search to be performed.

If a large number of examples of the parent object are to be found then all locations of sub-objects in the window are first determined before searching for consistent m-tuples of locations. Backtracking is used to discover the set of consistent m-tuples and this search may be speeded up using consistency and forward looking criteria as discussed above.

If the number of examples of the parent object sought for in the window is small (in relation to the anticipated existing number) then we adopt a different strategy in which we alternate between recursive search for new locations of sub-objects and backtracking search for a consistent allocation of found locations to sub-objects. At any instant, the constraint satisfaction process operates on a subset of found locations of each sub-object within the window. The procedure explores this space in an attempt to find a consistent allocation. If it fails, the next task is to search for more labels that may be assigned to the sub-objects. The selection of which sub-objects to search for, and the selection of sub-windows of the original window in which to search is done so as to maximize the probability of finding consistent allocations corresponding to locations of the parent object. Once new locations for some of the sub-objects have been found the constraint satisfaction procedure resumes on the augmented variable domains. The process oscillates between constraint satisfaction and the search for new sub-object locations till the desired number of consistent allocations corresponding to examples of the parent object are found, or the procedure announces failure.

We believe that the use of these two alternative strategies is an efficient way to accomplish spatial search. Studies involving this and other control issues will be presented in a forthcoming paper.

10. PRIMITIVE OBJECT SEARCH

The task given to the primitive search procedure is the determination of a specified number of locations of a primitive object. Each instance of the primitive object corresponds to a connected region in the search window. The primitive object is represented using a conjunctive normal form expression involving only PPROP properties. An example of such an expression is:

(((LAND X) (10 11)) \/ ((GEOL X) (1 2)))

/\ (((ELEV X) (50 90)) \/ ((ASPECT X) (30 40)))

The primitive object search procedure has two alternative strategies available, depending on the desired task:

a) To find a small number of individual instances of a primitive object it uses region growing by SEED EXPANSION.

b) To exhausively find examples of a primitive object in a window it uses region growing by CONNECTED COMPONENT LABELLING.

For each strategy the primitive object search procedure can also select a cutoff resolution level in the location tree database. At this resolution level all nodes are classified as either black or white.

Each node in the location tree database may be classified as WHITE, BLACK or GREY with respect to the primitive object the area of the node that satisfies the specified PPROP predicates.

Each node in the location tree has an area depending on its height in the tree. Let N denote the number of levels in the location tree. Then a node at level N, referred to as the lowest level, has a height of 0, and an area of 1 unit (pixel). A node at height H (level : N - H) has an area of $(2^H)^2$ units.

The selection of the area that must satisfy the predicates may be made using an absolute limit in the following manner. A node with an area of Y pixels may be considered BLACK only if it has more than (Y - X) pixels satisfying the predicates; GREY if it has between X and (Y - X) pixels satisfying the predicates; and WHITE if it has less than X pixels satisfying the predicates. This decision rule ensures that a node corresponding to a level with a node area of X units will be classified as only BLACK or WHITE preventing further descent of the tree by the region growing algorithm and fixing the resolution at the desired level. Such a rule enables the region growing procedure to take full advantage of compaction in the higher levels of the location trees and also

restricts the resolution to the desired level. Selecting X equal to 0 allows the search to be carried out at full resolution.

If a procedure wishes to view only a single level in the tree as in the case of a raster pyramid with no father-son links, then we may employ the following alternative rule. A node at some resolution level may be considered BLACK if more than X % of the area of the node satisfies the PPROP predicates specified, and WHITE if the area satisfying the predicates is between 0 and X %. Such a rule enables each layer to be viewed independently as a raster at the desired resolution.

The first step in the primitive object search procedure is the selection of an appropriate region growing strategy and an appropriate resolution level. The selection of strategy and resolution level is based on:

a) The desired number of examples.

b) The average size of the desired object in relation to the search window.

If the search strategy selected is SEED-EXPANSION then the constraint satisfaction procedure that calls the primitive object search procedure narrows the search window through the propogation of binary spatial relations (RPROPS) involving the primitive object and other sub-objects of the queried object that have already been searched for. In this way focus of attention is acheived in the calls to the primitive object search procedure, using RPROP constraint propogation. The first step in SEED-EXPANSION is a systematic search for an initial seed in the search window. This search is done using heuristic knowledge based on the size of the object, which is an indicator of the depth at which black nodes might be expected to occur. This heuristic is used to control the search for the seed, causing it to switch from a depth first search of the tree to a breadth first search at the selected depth. Once a seed has been found, it is grown using a SEED EXPANSION procedure, that finds the complete areal coverage of the region within the window. The procedure followed ensures that the maximal block representation of the region grown is returned. The procedure is iterated till the desired number of seeds have been grown.

Each node visited is tagged with a search tag, allowing the above procedure to systematically search for and grow seeds till the entire window, has been searched or the desired number of examples have been found.

If exhaustive search for the object is to be carried out then the CONNECTED COMPONENT LABELLING algorithm is applied to the search window. This is an application of the conventional blob coloring region growing algorithm using the quad tree data structure. The procedure is applied top down, marking BLACK nodes and merging connected components. The procedure descends to the next resolution level only when a GREY node is found and considers only the sons of the GREY node. In this way maximum use is made of the hierarchical tree structure of the location tree database. The procedure decends no futher then the appropriate resolution level where all nodes are classified as either BLACK or WHITE. All connected regions within the search window are returned by the procedure.

11. LEARNING

The main purpose of implementing learning procedures in KBGIS-II is to reduce query search time. It can be accomplished in to ways : either by remembering the results of previous search or by learning the definition of an object more precisely so that the search space may be pruned rapidly. Hence learning may be classified as either rote learning or inductive learning.

11.1. Rote Learning

Rote learning allows the system to memorize the examples of an object for which it has already searched, so that when it is asked to search for the same object again, it retrieves the previous examples instead of searching again. It stores only predefined high level objects.

Known examples are stored in a separate database that consists of a discrimination net for each defined spatial object. This database constitutes a part of the spatial object knowledge base. The discrimination nets used to store examples are basically pointer based quad trees. Each node in a discrimination net corresponds to a quad-tree window of the data base. Examples are stored

at the minimum containing block i.e. the lowest node which completely contains the example. Each object is stored in a different discrimination net. The data base also has one other discrimination net called the OBJECT-TREE that is used to store the name of the objects indexed by location. If the name of an object X is stored in a node Y, it implies that one or more examples the object X exist in the location tree database withim the quad-tree window corresponding to Y. This information is useful in answering queries of type (2).

A query for the locations of an object in a quad-tree window is answered by returning all examples stored in the sub-tree under the query node. If the low level search returns a new example of an object, it is added to the proper discrimination net and the OBJECT-TREE is also updated. Obviously, all found examples cannot be stored because the space requirement will increase monotonically. Hence, after finding a new example the system has to make a decision as to whether the example should be stored. The decision taken depends on various factors. If the complexity of an object is low, it can be searched for easily and therefore it is not stored in the object base. If an object is recursively defined in terms of other high level objects, a decision must be made as to whether the subobjects should be stored or the parent object. Again the decision taken depends on the cost of reconstructing the object from its subobjects. Besides the complexity, another criterion for storing the examples is the frequency with which they are sought.

There are two ways of storing the object, either exact locations or rectangle approximation. The rectangle approximation of an object can be represented by specifying its area, eccentricity, centroid and orientation.

11.2. Inductive Learning

Inductive learning is used to provide a new definition of an object from a given set of examples so that search for the object can proceed more efficiently.

To learn a new definition of an object either user can give input definitions or system itself can generate new definitions. Since it is not possible to include all possible PPROPS, RPROPS and GPROPS to give definitions, the user specifies the appropriate values and system generates

the definitions using those properties.

The inductive learning submodule of KBGIS-II is based on INDUCE[9]. INDUCE is a general purpose inductive learning program that takes a set of input rules and generates one or more output rules which are simpler, more general and consistent with the input rules. Given a set of input rules, it first finds a set of alternative consistent generalizations by locating the most promising clauses (which are common) and adding new clauses to each of them until a set of consistent generalizations of the event is obtained. After getting the cover, it extends the response of the functions and then selects the best generalization from this set and removes the rules for which this is a generalization. The criteria for selecting the best rules as well as the number of the output rules can be changed by varying parameters.

INDUCE has the facility of providing background knowledge and the user can add arithmetic and logic rules for generalization. Besides background rules INDUCE also has the capability of adding new functions, equivalence predicates and extremity predicates.

The language of INDUCE is different from that of KBBGIS-II. Hence translators are used to convert an object definition from one language to other. First we discuss the KBGIS-II-INDUCE translator and then INDUCE-KBGIS-II translator.

11.2.1. KBGIS-II - INDUCE

This translator takes a set of rules in KBGIS-II language and converts it into INDUCE format. Besides the syntax transformations, it performs the following tasks:

a) The current implementation of INDUCE does not allow disjunctions, therefore if an input rule has a disjunction, it splits the rule into two rules, e.g.

$$(X_1 \vee X_2) \wedge Y \Rightarrow [d=1]$$

becomes

$$X_1 \Rightarrow [d=1]$$

$$X_2 \Rightarrow [d=1]$$

In KBGIS-II GPROPS and RPROPS do not have disjunctions but PPROPs can have a clause which has disjunction of two layers. Hence, for each disjunction it generates a new rule, i.e. for the following input rule

((TYPE O_1) LAKE)

/\ (((LAND O_2) 21) \/ ((GEOL O_2) 22))

/\ (((ELEV O_2) (100 200)) \/ ((SLOPE O_2) (20 40)))

the output will be

[TYPE (O_1) = LAKE][LAND (O_2) = 21][ELEV (O_2) = 100..200] => [d=1]

[TYPE (O_1) = LAKE][GEOL (O_2) = 21][ELEV (O_2) = 100..200] => [d=1]

[TYPE (O_1) = LAKE][LAND (O_2) = 21][SLOPE (O_2) = 100..200] => [d=1]

[TYPE (O_1) = LAKE][GEOL (O_2) = 21][SLOPE (O_2) = 100..200] => [d=1]

b) INDUCE cannot handle real numbers and also the range of values should not be very large (typically < 100) therefore values of GPROPS and RPROPS are properly normalized.

11.2.2. INDUCE-KBGIS

This translator takes a set of rules in INDUCE language and converts it into KBGIS language. In the current implementation the language of KBGIS is not fully compatible with the language of INDUCE, therefore if there is any input clause that cannot be converted into KBGIS language it is ignored.

If the system has learnt the definition of a new object, it is directly stored in the Spatial Object Knowledge Base. Otherwise, the system compares the new definition with the old one and if it is better the Spatial Object Knowledge Base is modified. In the current implementation due to language incompatibilities, sometimes the system may not be able to handle the INDUCE output. In such cases the user may interpret the output and update the Knowledge Base, using the Knowledge Base Editor.

12. SUMMARY AND CONCLUSIONS

KBGIS-II, as described in this essay, is currently implemented and running on a VAX-11/750 under the VMS operating system at the University of California, Santa Barbara. The system is programmed in Common Lisp, Pascal and C. We now briefly summarize the degree to which KBGIS-II meets the four requirements, laid out above, and the manner in which the four general principles, also listed above, are used to meet these requirements. It should be emphasized that the properties of the currently-implemented system are still under investigation and that there are plans to continue development of KBGIS-II. We therefore discuss both current research that is being performed using the current system and planned extensions to the system.

12.1. Reqirements and Principles.

The system is currently capable of handling large, multilayered, heterogeneous, spatially-indexed databases. The software design entailed by the overall system requirements described below has of necessity made the current hardware (VAX 11-750) a limiting factor in the size of the databases that can be handled at interactive speeds The transfer of the system to more appropriate hardware (such as a LISP machine) would resolve much of this problem. The system has the capability of responding to all the queries of types (1) and (2) that are expressible in our spatial object language (SOL). Although research is still in progress on the matter, the processing of the queries appears to be relatively efficient in the sense of reducing the average complexity of the search for spatial objects. The hardware deficiencies, however, do not permit the system to be truly interactive in the case of queries concerning complex spatial objects. KBGIS-II is flexible with respect to both domains of application and users.

Concerning the role of the four sets of principles in allowing the system to satisfy these four requirements, we make the following comments:

a) The development of the system suffered from a failure to adhere to the principled use of the techniques of software engineering, although it benefitted from the systemmatic aplication of techniques from database management (in the construction and storage of the location tree

database, where the spatial image data is segmented into retreivable areas that are paged in on demand, see Klinger[12]); from the use of the theory of algorithms and complexity (in the construction of spatial search procedures); from the use of AI techniques (in the structuring of the knowledge base, in the design of the spatial search procedures and in the application of the learning procedures); and from the application of computer graphics techniques (in terms of the system output).

b) The integration of techniques from computer vision and image processing provide the system with an ability to handle queries of a type not typically found in GIS, while allowing the system to integrate both image and digital cartogrtaphic data.

c) The six priciples discussed in the general section on search greatly reduce the computational effort of the system in responding to queries, as compared with standard, exhaustive raster-based search procedures.

d) Finally the availabilty of various editors allows the system to be easily tailored for use in various spatial domains and for various users.

12.2. Investigation of System Performance

Investigations of the system's ability to handle various queries concerning a large geological database are currently underway, and will be reported in future publications. The main research effort involves an empirical analysis of the efficiency of the search procedures, and of the effects of varying various parameters that affect the efficiency of search.

12.3. Extensions to the system

Planning is currently underway concerning modifications to KBGIS-II that will both improve the efficiency of its current processing capabilities and extend its current capabilities. The planned extensions include:

a) Adding computer cartographic capabilities and ordinary polygon processing functions that are similar to those found in such currently available systems, such as ARC/INFO.

b) Adding "fuzzy" spatial object definitions and "fuzzy" reasoning.

c) Adding a database and specialized processing functions for remotely-sensed data and an interface between this database and KBGIS-II; adding procedures for map-guided image interpretation; and providing data structures and procedures that permit joint querying of both the digitized cartographic and image databases.

d) Providing the system with the capability of answering a class of queries that involve detection of change over time.

e) Providing procedures and control structures that permit the inductive learning procedures of the system to operate autonomously.

ACKNOWLEDGEMENTS

We would like to acknowledge Ted Albert of USGS for his continued support of the project and UCSB graduate students Kam Chow, Yuan Lui, Zhan Xi-Chang and Micha Pazner for their help in developing the earlier versions of the system.

References

1. Aronson, P., "Applying Software Engineering to a General Purpose GIS," *Proceedings Auto-Carto 7, American Society of Photogrammetry*, pp. 23-31, Washington, 1985.

2. Ballard, D.H. and Brown, C. M., *Computer Vision*, Prentice Hall, Englewood Cliffs, N.J., 1982.

3. Caulkins, H. W., *A Pragmatic Approach to GIS Design*, IGU Commission on Geographical Data Sensing and Processing, New York, 1983. and J. O'Callaghan

4. Charniak, E., Riesbeck, C. K., and McDermott, D.V., *Artificial Intelligence Programming*, Lawrence Erlbaum, Hillsdale,N.J.., 1980.

5. Charniak, E. and McDermott, D., *Introduction to Artificial Intelligence*, Addison-Wesley, Reading, Massachussetts, 1985.

6. Haralick, R.M. and Shapiro, L.G., "The Consistent Labelling Problem : Part I," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-I, pp. 173-183, 1979.

7. Haralick, R.M. and Shapiro, L.G., "The Consistent Labelling Problem : Part II," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-2, pp. 193-203, 1980.

8. Haralick, R.M. and Elliott, G.L., "Increasing Tree Search Efficiency for Constraint Satisfaction Problems," *Artificial Intelligence*, vol. 14, pp. 263-313, 1980.

9. Hoff W., Michalski R. S., and Stepp R., "INDUCE/2: A Program for Learning Structural Descriptions from Examples," UIUCDCS-F-83-904, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, 1983.

10. Hunter, G.M. and Steiglitz, K, "Operations on Images using Quad Trees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, pp. 145-153, 1979.

11. Jackson, M. J., *The Development of Integrated Geo-Information Systems*, Reading, England, 1985. Paper presented at "survey and Mapping, 1985"

12. Klinger, A. and Rhodes, M.L., "Organization and Access of Image Data by Areas," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, pp. 50-60, 1979.

13. Mackworth, A.K., "Consistency in Networks of Relations," *Artificial Intelligence*, vol. 8, pp. 99-118, 1977.

14. Marble, D. F., "On the Application of Software Engineering Technology to the Development of Geographic Information Systems," *Workshop on Design and Implementation of Computer Based Geographic Information Systems*, Honolulu, 1982.

15. Mackworth, A.K., "The Complexity of some Polynomial Network Constraint Algorithms for Constraint Satisfaction Problems," *Artificial Intelligence*, vol. 25, pp. 65-74, 1985.

16. Nilsson, N.J., *Artificial Intelligence*, Tioga Press, Palo Alto, Ca., 1980.

17. Peuquet, D. P., "A Conceptual Framework and Comparison of Spatial Data Models," *Cartographica*, vol. 21, pp. 66-113, 1984.

18. Preparata, F. P. and Shamos, M. I., *Computational Geometry*, Springer-Verlag, New York, 1985.

19. Samet, H., "Region representation : Quadtrees from binary arrays," *Computer Vision, Graphics and Image Processing*, vol. 18, pp. 88-93, 1980.

20. Samet, H., "An algorithm for converting rasters to quadtrees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 3, pp. 93-95, 1981.

21. Samet, H., "Connected component labelling using quadtrees," *JACM*, vol. 28, pp. 487-501, 1981.

22. Samet, H., "The Quadtree and Related Hierarchical Data Structures," *ACM Computing Surveys*, vol. 16, pp. 187-260, 1981.

23. Smith, T. R. and Peuquet, D. J., *in a Knowledge-Based Geographic Information System*, London, England, 1985.

24. Tanimoto, S. and Pavlidis, T., "A Hierarchical Data Structure for Picture Processing," *Computer Graphics and Image Processing*, vol. 4, pp. 104-119, 1975.

Figure 1
The Architecture of
KBGIS II

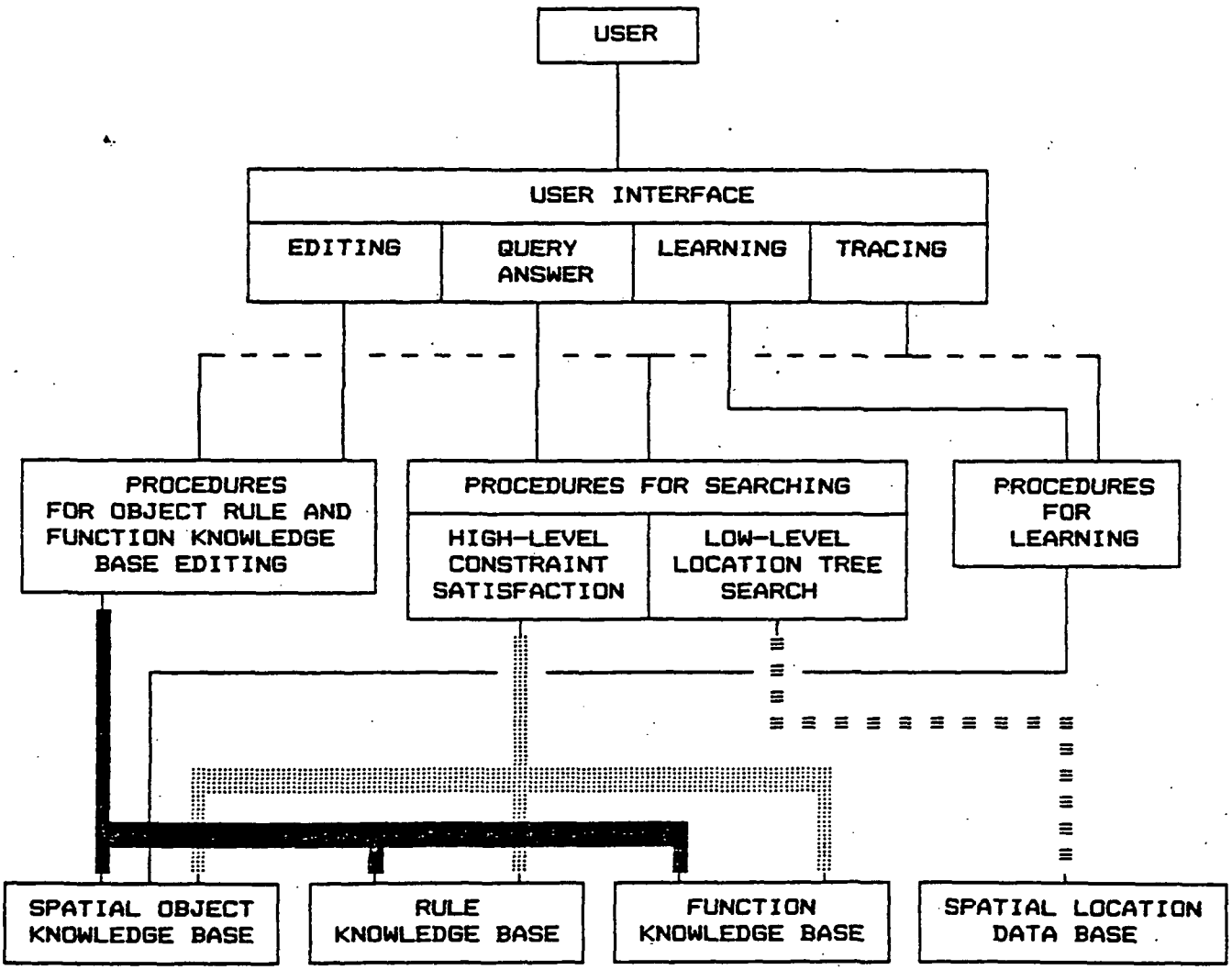


Figure 2 The Spatial Object Data Structure

SLOT-NAME	CONTENTS
FeatureType	Distinguishes between linear and region based features.
DefinedBy	The definition of the object, a disjunctive normal form expression in the spatial object language.
Defines	A list of heirarchically higher objects that are defined in terms of the object.
Heuristics	Other objects whose locations are contextually related to the locations of the object, together with the nature of the spatial relation involved.
Complexity	A measure of the complexity of search for new examples of the object.
Size	An approximation to the linear and areal dimension of the object.
Procedures	Pointers to low level algorithms that can operate directly on the image without recourse to the definition of the object.

Figure: 3 The Function Data Structure

SLOT-NAME	CONTENTS
Propogation	Indicates, in the case of GPROPS and RPROPS, if the function can be inverted to propogate constraints during search.
Complexity	A measure of the computational complexity of the function, used in the calculation of the complexity of spatial objects defined using the corresponding GPROP or RPROP.
Symmetry	Information on the symmetric properties of binary (RPROP) functions, permitting arguments to be switched by the search control, depending on dynamic object prioritization.
Range	The nature of the values that the user may specify as desired when the corresponding GPROP or RPROP is used to define an object.
Subroutines	The names of subroutines called by the function, used in system management.

Figure 4