University of California
Los Angeles

Knowledge-Based Image Retrieval Using Spatial and Temporal Constructs

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Chih-Cheng Hsu

1997

The dissertation of Chih-Cheng Hsu is approved.

Lee Cooper

D. Stott Parker

Alfonso Cardenas

Wesley W. Chu, Committee Chair

University of California, Los Angeles

1997

To my parents and Peijung

Table of Contents

List of Figures

List of Tables

Acknowledgments

acknowledgments

Vita

1988	B.S. (Computer Science), National University of Taiwan, Taipei, Taiwan
1988–1990	Second Lieutenant, Computing Center, Chinese Military Academy, Fengshan, Kaoshiung
1991	M.S. (Computer Science), State University of New York at Stony Brook, Stony Brook, New York
1992–1997	Research Assistant, Computer Science Department, UCLA.

Publications

Wesley W. Chu, Kuorong Chiang, Chih-Cheng Hsu, Henrick Yao, "An Error-based Conceptual Clustering Method for Providing Approximate Query Answers", *Communications of ACM, Virtual Extension Edition (http://www.acm.org/cacm/extension)*, December, 1996

Chih-Cheng Hsu, Wesley W. Chu, Ricky K. Taira, "A Knowledge-Based Approach for Retrieving Images by Content", *IEEE Transactions on Knowledge and Data Engineering*, August, 1996

Wesley W. Chu, Chih-Cheng Hsu, Ion Tim Ieong, Ricky K. Taira, "Content-Based Image Retrieval Using Metadata and Relaxation Techniques", book chapter in *Managing Multimedia Data: Using Metadata to Integrate and Apply Digital Data*, edited by Wolfgang Klas and Amit Sheth, Prentice-Hall Publishing, 1998

Chih-Cheng Hsu, Wesley W. Chu "A Knowledge-Based Object-Oriented Language for Temporal Image Databases", *Technical Report, UCLA*, December, 1997

Abstract of the Dissertation

Knowledge-Based Image Retrieval Using Spatial and Temporal Constructs

by

Chih-Cheng Hsu

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 1997

Professor Wesley W. Chu, Chair

A knowledge-based approach is introduced for retrieving images by content using spatial and temporal constructs. It supports the answering of conceptual image queries involving similar-to predicates, spatial semantic operators, and references to conceptual terms, as well as temporal, evolutionary, and stream constructs. Interested objects in the images are represented by contours segmented from images. Image content such as shapes and spatial relationships are derived from object contours according to domain-specific image knowledge. Sequences of image objects are represented as streams for retrieving image (sequences) based on their temporal change.

A three-layered model is proposed for integrating image representations, extracted image features, and image semantics. With such a model, images can be retrieved based on the features and content specified in the queries.

A knowledge-based spatial temporal query language (KSTL) is also presented to express and process image queries with conceptual, spatial, temporal, evolu-

tionary, and stream constructs. The implementation of KSTL via extending ODMG's object-oriented query language OQL [Car94] is also presented.

The knowledge-based query processing is based on a *query relaxation* technique. The image features are classified by an automatic clustering algorithm and represented by Type Abstraction Hierarchies (TAHs) for knowledge-based query processing. Since the features selected for TAH generation are based on context and user profile, and the TAHs can be generated automatically by a clustering algorithm from the feature database, the proposed image retrieval approach is scalable and context-sensitive.

CHAPTER 1

Introduction

In image databases, image content can be queried based on either pixel-level or object-level information [GJ97]. Pixel-level information such as color distribution, histograms, and layout of color blocks can be used to query image content [NBE93]. However, the semantic information of objects in images cannot be properly represented at pixel-level.

In many image database applications, such as medicine and geographical information systems (GIS), image objects are spatial, temporal, and evolutionary. The spatial characteristics of objects such as shape, location, and spatial relationships among objects are important information to be modeled for querying image content at object level. Objects in images can evolve from one stage into others (e.g., a tadpole evolves into a frog), or one object may split into several objects (e.g., one brain tumor splits into several). Temporally related image object snapshots can be represented as *streams*. Using streams as a queryable unit to express temporal constraints improves the temporal expressiveness of queries when compared with other temporal languages without stream constructs [Sno87].

In feature-based querying, the precise specification of feature values are often very difficult if not impossible. Using semantic operators (such as *similar_to*, *near_by*), conceptual terms (such as *large*, *heavy*), and approximate operator mimics human cognitive process and expression, and thus greatly eases query constraint specification [HCT96, HC97, BPJ93].

To store a vast amount of images in image databases as well as effectively and efficiently support the conceptual, spatial, temporal, evolutionary, and stream constructs to query image content, in this dissertation, we propose the following techniques to address these issues:

1. A multi-layered knowledge and data model to integrate raw images, extracted image features, semantic objects, and knowledge. The model provides a scalable and extensible architecture for knowledge-based image query answering and data storage.
2. Modeling constructs to represent spatial, temporal, evolutionary, and stream information of images,
3. Techniques to support similarity retrieval based on a set of specified spatial features of image (objects),
4. A knowledge-based query language, KSTL, which supports spatial, temporal, evolutionary, and stream queries
5. Techniques to process the proposed query language

1.1 Related Works and Contributions of the Dissertation

Comparisons among the proposed and related techniques are presented as follows to highlight the contributions of this dissertation work.

1.1.1 A Multi-Layered Knowledge and Data Model

Currently, images cannot be easily or effectively retrieved due to the lack of a comprehensive data model that captures the structured abstracts and knowledge needed for image retrieval. To remedy such shortcomings, we propose a multi-layered image model which integrates raw image representations, semantic representation of image objects, and knowledge representations of image features into one inter-referenced multi-layered model. The model consists of raw image layer (RL), semantic layer (SL), and knowledge layer (KL). A similar multi-layered model, VIMSYS [GWJ91], lacks the knowledge representation provided in our image model.

Data within each layer are represented and stored using the physical and logical schema that are of sufficient expressiveness and simpler maintenance for data in a layer. The separated logical design for each layer makes the image databases scalable and extensible since the logical model chosen for a layer must be extensible to incorporate any new data and knowledge. For example, object-oriented model is used in our semantic layer which allow additional semantic objects and their relationships with existing objects to be added and maintained. Type abstraction hierarchies [CYC96] and stack model [CCK84] are the logical models used in the knowledge layer and representation layer, which are both extensible and scalable. Optimization and indexing are performed based on the corresponding physical storage schema in each layer to provide efficient information retrieval in each layer. The logical schema in each layer allow inter-layered linking on related information. The temporal, evolutionary, and spatial objects are integrated in the semantic layer to inter-link with related raw-images and knowledge to allow intra-layer and inter-layer computation and query retrieval. Thus the schema browser, query language, and query processor can reference and/or retrieval all related objects, images, and knowledge automatically through the inter-layered links/mapping during the data insertion and update, query specification and processing..

1.1.2 Spatial Content Modeling

Pixel matching methods employed for content-based retrieval are time-consuming and of limited practical use since little of the image object semantics is explicitly modeled. Object-level image content querying uses the characteristics of objects and their temporal and spatial information as query constraints to retrieve images. Global shape features used to retrieve similarly shaped objects (such as area and circularity used in QBIC [NBE93]) are limited due to its limitation on representing the shape of complex object [Lev85]. Shape of complex-shaped objects need to be modeled in order to query its shape content. A landmark-based decomposition technique is proposed in the dissertation to represent the shape of complexed shaped objects. After decomposing a complexed shaped object into simpler components based on well-recognized landmark(s), specific shape features of the objects can be represented and queried based on the shape and spatial relationships of the decomposed simpler shapes.

Spatial relationships among objects are important image content for querying. Chang et al. [CYD88] models the distribution of image objects using orthogonal spatial relationships (e.g., north of, south of). Chu et al. [CIT94] models both the orthogonal and topological (e.g., contained, separated) spatial relationships. These qualitative modeling approaches provide high level description of spatial relationships which is easy to understand and query. However, the approaches lack the preciseness when compared with quantitative description of the spatial relationship using spatial relationship features such as the shortest distance or ratio of overlapping among objects. Depending on the application domains and data distribution, either qualitative or quantitative approach could be preferred. We propose an *ontology-based quantitative approach* to model spatial relationships content [HCT96, CHI98]. Values of image features are extracted from images to allow direct quantitative querying on spatial relationship features. The semantic terms in the ontology of interested spatial relationship (e.g., nearby, invading, north of) are defined as the value ranges of spatial relationship features under the guidance of domain experts based on the clustering information discovered from database instances [CCH96]. Thus, in addition to directly querying the numerical values, semantic terms can also be used to query spatial relationship in our approach. Our ontology-based approach provides easy to understand spatial relationship description and querying without sacrificing the precision.

1.1.3 Temporal, Evolutionary, and Stream Modeling

Temporal database research [Sno87] has been focused on comparing the temporal relationships of objects and intervals using constructs such as BEFORE and AFTER. Evolutionary querying proposed in [CIT94] allows constructs regarding the class changes of objects (i.e., evolution, fusion, and fission) to be queried. Temporally related image object snapshots can be ordered and represented as *streams* [DC96]. Streams can be queried and retrieved based on specific values of individual snapshots or the trend of value changes of object features. Using streams as as a queryable unit to express temporal constraints improves the temporal expressiveness of queries when compared with other temporal languages without stream constructs [Sno87]. A corresponding query language is proposed to use streams to express complex temporal constraints. Temporal logic is used as the formalism to verify the modeling and querying [GM91, HC97].

1.1.4 Intelligent Retrieval

To support conceptual terms (large, heavy) and similar-to and semantic operators (e.g., nearby, far_away) in image queries, knowledge-based approach is required to resolve these conceptual constructs for intelligent retrieval. VIMS [BPJ93] retrieves similar images by relaxing feature values of the target image based on standard deviation of features. Independent of the target data values, the same amount of relaxation is applied on the target data values to represent the similarity of data. Such interpretation of similarity is not sensitive to the location of the target data values inside their value range. Moreover, in an image data space, many features are based on multiple attributes. For example, *location* requires at least two attributes (i.e., positions on x-axis and y-axis). Using a standard deviation to interpret the variation of multi-attribute features lacks the consideration of correlation among different attributes. In this dissertation, an instance-based knowledge discovery technique, MDISC [CCH96], is used to cluster similar images based on the user-specified image features (e.g., shape descriptors and spatial relationship features). The knowledge required to resolve the conceptual terms and *similar-to* and semantic operators is defined by domain experts based on the clustering information generated by MDISC clustering algorithm. MDISC, as a conceptual clustering algorithm, clusters instances based on values of a specified set of features. MDISC provides semantic clustering information which is different from other multi-dimensional indexing techniques, such as K-D-B-tree (used in FIBSSR [MG95]) and R^* tree (used in QBIC [NBE93]). MDISC classifies images based on conceptual difference of the feature values, while K-D-B-tree and R^* tree cluster data based on minimizing the number of disk access per data retrieval. The K-D-B-tree and R^* tree clustering techniques do not consider the semantic difference of image features; thus no global conceptual view of the image clustering can be provided to represent conceptual predicates such as `LARGE` *tumor* and *tumor* `NEARBY` *an organ.*

1.1.5 A Knowledge-Based Spatial Temporal Query Language

Objects in images are *spatial, temporal, and evolutionary.* Temporally related image objects are represented as *streams.* A query language supporting spatial, temporal, evolutionary, and stream constructs is required. Further, since the exact value of image features is difficult to specify by users, semantic operators (e.g., *similar_to*, *near_by*), conceptual terms (e.g., *large*, *heavy*), and value approximation (e.g., *about* 10 mm) are useful in the constraint specification. Existing query languages [RFS88, OM88, HCT96] do not integrate this query capability into an integrated query language for querying image content. To remedy these shortcomings, we propose a Knowledge-based Spatial Temporal Query Language (KSTL) for image databases that is capable of expressing conceptual, spatial, temporal, evolutionary, and stream queries. The implementation of KSTL is based on extending ODMG's object-oriented query language OQL in order to leverage on available commercial products and standards [Car94].

1.1.6 Knowledge-Based Query Processing

During the query processing, the proposed conceptual constructs and extended functions/methods for temporal, evolutionary, and stream behaviors should be converted into commands that can be processed by commercially available OQL query processor [Car94]. Table 6.1 categorizes the extended OQL constructs to be implemented for KSTL.

Data in the image databases are organized into class hierarchies based on their data structures and required computational and querying functionalities. Common property such as spatial, temporal, evolutionary, and stream property can be implemented in the top level superclasses of the class hierarchies. These top level superclasses are called *kernel-classes*. We use a kernel class-based approach to implement the extended functions/methods. Thus, the commercially available OQL query processor [Car94] can invoke the methods and functions of the implemented kernel classes to directly process the KSTL query. On the other hand, the conceptual constructs in KSTL need to be converted into OQL query processor. A knowledge-based query modification technique is used [CYC96, HCT96] to convert the conceptual constructs in KSTL into specific value ranges of image features based on the related knowledge in the knowledge layer. These converted value ranges of image features are used directly by the query processing module to retrieve data and image instances from the database.

CHAPTER 2

Multi-Layered Knowledge and Data Model

A multi-layered knowledge and data model is proposed in this chapter to provide a scalable and extensible architecture for image databases. Image content can be divided into different levels of abstraction (e.g., pixel values of raw image and the contour size of an image object are data of different abstraction level). Data of the same abstraction can be modeled and stored using the same logical and physical schema. The logical schema (e.g., object-oriented model) for knowledge or data of the same data abstraction level is used to organize, query, and compute the data that it represents. The corresponding physical schema provides the required indexing structures and compression strategy to optimize the storage or retrieval efficiency. Related representations among different layers are inter-linked to allow intra- and inter-layered feature computation, knowledge reference, image display, and data retrieval. The multi-layered image model makes the proposed system scalable and extensible.

The proposed multi-layered model consists of the Representation Layer (RL), the Semantic Layer (SL), and the Knowledge Layer (KL) (see Figure 2.1).

Representation Layer Raw images and contours of objects in the images are stored in the RL. Related images are organized and stored in various stacks for efficient access [CCK84]. The RL also includes: 1) characteristics of the acquired raw images and 2) contours of queryable objects in the images. The characteristics of raw images includes information such as pixel width, dimensionality of images, the height and angle that a brain MR image is taken, number of slices in an MR image set, the spacing between two MR image slices, the machine and technician acquiring the images. The contours of queryable objects can be segmented manually, semi-automatically (e.g., using techniques like snake [Dv93] and flooding in [NBE93]), or automatically [WLK92, WA95] depending on the contrast and separability of the image objects.

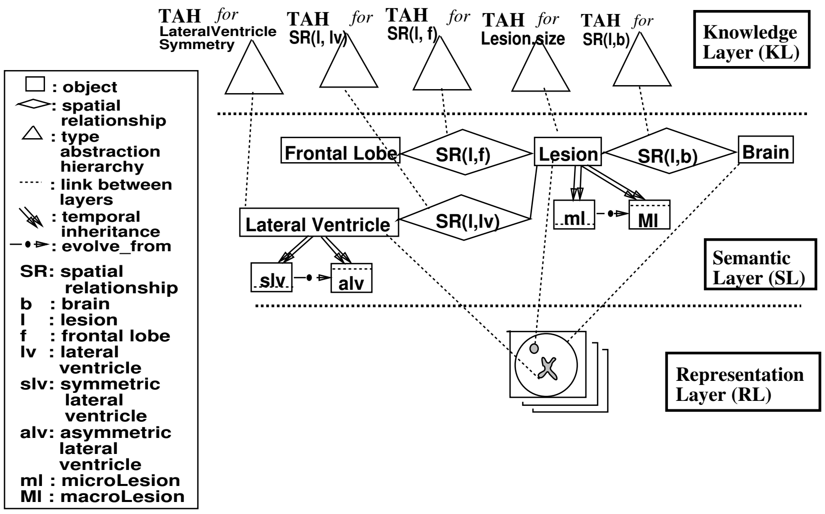


Figure 2.1: Modeling of image features and content using multi-layered architecture

Semantic Layer In SL, semantic objects are modeled using object-oriented approach to represent the features extracted from the contours of interested objects. The extracted features such as shape (e.g., circular, thick), spatial relationships (e.g., *Joined, Disjoined*, etc.) and evolutionary relationships (e.g., *Evolve, Split, Fuse*) are modeled as either object classes or attributes. Common features and class characteristics are placed in the superclasses inside the class hierarchy.

Knowledge layer The knowledge layer (KL) consists of the required knowledge structures to support approximate and conceptual query answering on image content. An instance-based conceptual clustering technique, MDISC [CCH96], is used to cluster image objects based on a set of specific features corresponding to the query context and user types. The clustering of image objects is represented in a hierarchical knowledge structure, called Type Abstraction Hierarchy (TAH) [CCH96], to resolve the conceptual terms (e.g., *large* and *heavy*) and similar-to and semantic operators (e.g., *nearby* and *far_away*) in KSTL.

The queryable image objects are represented as semantic objects in SL. Since the related knowledge of a semantic object are inter-linked, the conceptual constructs of the objects appearing in KSTL can be resolved based on the referenced knowledge structure in the KSTL. Inter-linked raw images can then be displayed as part of the query results. Additional computation on the corresponding raw images can also be accomplished via the link. The multi-layered image content representation is maintained during the data update and insertion.

CHAPTER 3

Feature-Based Image Content Modeling

Features of image objects can be used to index images, and the image retrieval is based on the indexed features [GM90]. The information retrieval approach [RS91, RS92] transforms image and a query into signatures, and the query is then answered by matching the signatures. There are approaches which store boundaries of image objects using minimum bounding box representations [RFS88, GWJ91]. Since a bounding box is merely an approximation of the boundary of the original image object, representing the image object with such structures may not allow the user to retrieve all the answers that satisfy the query specification. Spatial relationships among the image objects are important features for image content querying. The orthogonal relationships among objects can be encoded into a 2D string [CYD88]. As a result, user queries concerning the orthogonal relations of the objects can be answered.

In addition, there are attempts to answer approximate queries concerning spatial objects. QBIC [NBE93] uses object features such as *area* and *circularity* to retrieve similarly shaped objects. VIMS [BPJ93] applies standard deviation on simple spatial features, such as eyebrow width, to retrieve similarly shaped (face) images. In this chapter, we propose a landmark-based decomposition technique to provide specific shape features for complex-shaped image objects. An ontology-based quantitative approach is also proposed to model spatial relationships. The approach provides more precise spatial relationship description than 2D strings while still allows semantic terms (e.g., nearby, north of) to be used in querying spatial relationships.

Temporal database research [Sno87] has been focused on comparing the temporal relationships using constructs such as BEFORE and AFTER. Evolutionary constructs proposed in [CIT94] allows class changes of objects to be modeled and queried. Temporally related image object snapshots can be represented as *streams*. Stream constructs allow the trend of value changes of object features to be tracked. Using streams as a queryable unit improves the expressiveness of temporal queries when compared with other temporal languages without stream constructs [Sno87]. Modeling of spatial content is presented in Section 3.1 and modeling of temporal, evolutionary, and stream behavior of objects is presented in Section 3.2.

3.1 Spatial Content Extraction and Modeling

3.1.1 Extracting Shape Features Via Decomposition

Shape is an important characteristics of image objects. Global shape measurements such as *area*, *perimeter*, and *circularity* used in QBIC[NBE93] and *curveness*, *rectangularity*, *compactness*, *direction*, *elongatedness*, and *eccentricity* [SHB93] provide approximate shape measures for contours of relatively simple shapes (*i.e.*, shape without irregular intrusions and/or protrusions). However, for contours of complex shapes (i.e., shapes with irregular intrusions and/or protrusions), these global shape descriptors do not provide sufficient expressive power for describing their specific shape content [Lev85]. Therefore, we propose a *landmark-based decomposition technique* to decompose complex-shaped contours into shape descriptive sub-structures based on the content-related landmark(s) in the images.

Landmarks are characteristic points in an image which are easily recognizable on all images and can be used to decompose *every* instance of the complex object into descriptive sub-structures (e.g., the tips and webs of a hand are the landmarks for decomposing a hand contour for detailed shape description of a hand). Landmark points in a contoured object can be obtained, for example, by matching model segments with corresponding data segments as described in [Gri90, LTS95].

With the landmark identified, the complex-shaped contours are decomposed into *shape-descriptive* sub-structures. The shape content of the complex-shaped object can be represented by the simpler shape and spatial relationships of the decomposed sub-structures. Attributes of the sub-structures can be common spatial characteristics (such as area, length, and width) or shape descriptors of the decomposed sub-structures. Capturing these attributes provides a more precise description on the shape of the image objects.

The lateral ventricle is used as an example object to illustrate landmark-based decomposition in studying the shape symmetry of the lateral ventricle. In medicine, the symmetry of the lateral ventricle is observed using the two tip notches of the brain contour as the landmarks. The brain contour is modeled using the *shape contour model* to automatically identify the two tips [Gri90, LTS95]. Then the lateral ventricle can be decomposed into four substructures as shown in Figure 3.1 to studying the symmetry of lateral ventricle. We can express the symmetry aspect of the lateral ventricle via the relative shape among these de-

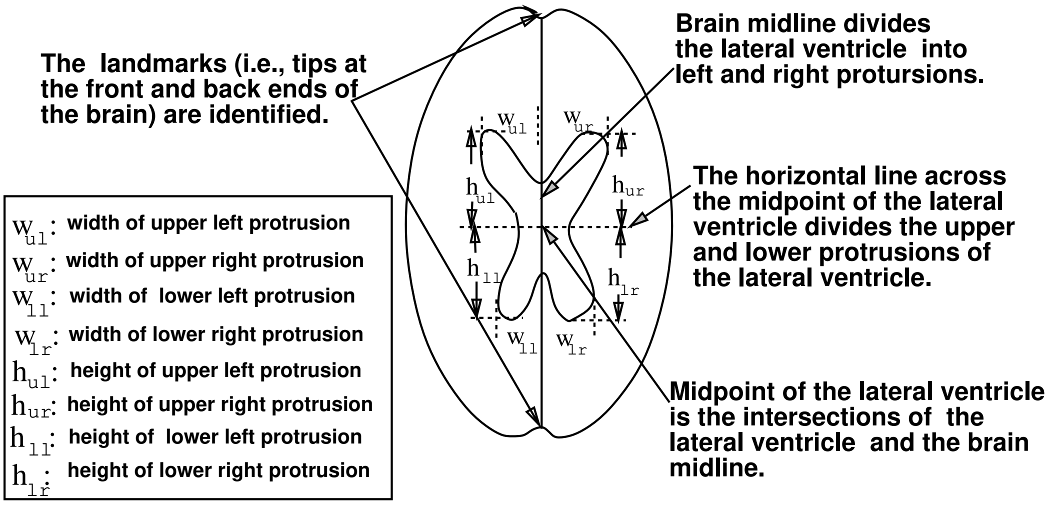


Figure 3.1: For more precise shape description, the shape model decomposes a lateral ventricle into four natural sub-structures: upper left substructure, upper right substructure, lower left substructure, and lower right substructure.

composed contours. In Figure 3.1, the height and width of the four components of a lateral ventricle are used to construct a *shape feature* to describe the left to right symmetry of the lateral ventricle as (upperLRWidthRatio (w_{ul}/w_{ur}), upperLRHeightRatio (h_{ul}/h_{ur}), lowerLRWidthRatio (w_{ll}/w_{lr}), lowerLRHeightRatio (h_{ll}/h_{lr})). The *left to right symmetry* of the lateral ventricle is an important shape feature for detecting various brain diseases [DSB95].

The proposed landmark-based decomposition technique provides an effective quantitative shape description when the image objects have limited numbers of shape components. The decomposed objects provides sufficient image content to retrieve specifically or similarly shaped image objects. Further, with image classification algorithms, such as MDISC [CCH96], conceptual terms can be defined to describe a shape feature. The *shape feature description table* (Table 3.1) lists the conceptual terms for describing the shape features. Thus, users can use

object feature	**conceptual terms**
lesion.size	*small, medium, large*
lesion.roundness	*circular, non_circular*
lesion.surface	*smooth, coarse*
lateral_ventricle.left_to_right_symmetry	*upper_protrusion_pressed_to_the_right,* *upper_protrusion_pressed_to_the_left*, ...
...	**...**

Table 3.1: A *shape feature description table* for the brain

conceptual terms for describing a specific shape in a query: for example, *"retrieving images with lateral ventricles whose upper protrusion are pressed to the right"* (the MTAH defining *upper protrusion pressed to the right* is shown in Figure 3.2).

3.1.2 Extracting Spatial Relationship Features

Qualitative description of spatial relationship (e.g., north of, south of) provides easy-to-understand high level description of spatial relationship [CYD88, CIT94]. However, the approach lacks the preciseness if compared with quantitative description of the spatial relationship. Depending on the application domains and data distribution, either qualitative or quantitative approach could be preferred. An ontology-based quantitative approach is introduced here to allow both quantitative and qualitative description and querying of spatial relationships of image content.

Modeling spatial relationships merely by simple semantic constructs such as *separated* and *connected* is insufficient to express real-life spatial relationships (as illustrated in Figure 3.3). Additional parameters are needed to describe the spatial relationships more precisely. A set of required spatial relationship features should be specified by domain experts. The primary concerns for a quantitative description of spatial relationships include:

- Containment relationships which describe the relative position and locations of contact between a pair of objects. The detailed containment characteristics are further quantitatively captured by features such as ratio of overlap, ratio of area, the shortest distance between their contours, etc.

- Directional relationships describe the directional characteristics between ob-

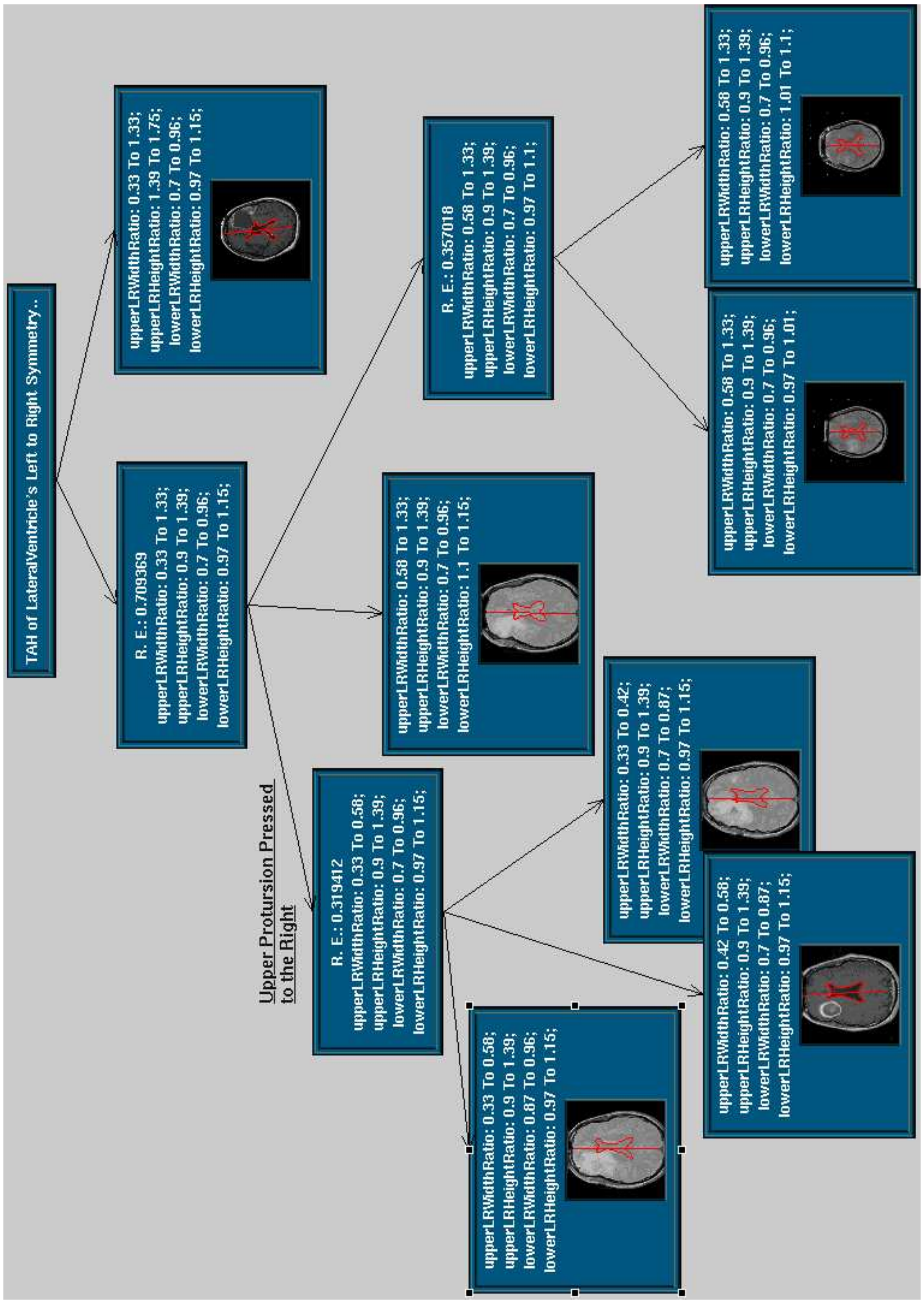


Figure 3.2: Multi-attribute type abstraction hierarchy (generated by MDISC based on the decomposed four protrusions) representing the left to right symmetry of the lateral ventricles

jects. These directional features can be quantitatively captured by features such as relative distance of centroid on *x axis, y axis*, and *z axis* respectively.

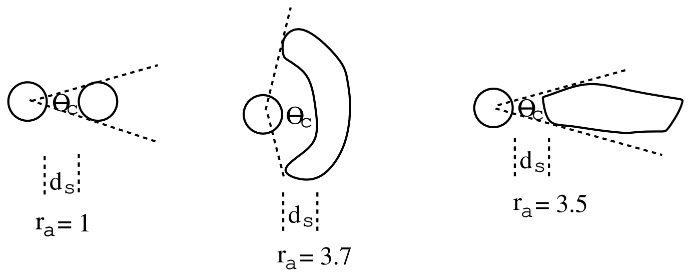


Figure 3.3: An example showing that merely using semantic operators (e.g., *SEPARATED*) and/or single measurement (e.g., the *shortest distance* (d_s)) is insufficient to capture the spatial relationship of two objects. We need additional features such as *angle of coverage* (θ_c) and *ratio of area* (r_a) to classify the illustrated spatial relationship.

Based on anatomical knowledge and physician diagnosis, the spatial relationships of an object with respect to its surrounding objects can be derived; we shall call them the *primary spatial relationships* of the objects. We use primary spatial relationships to represent the semantics of the images in the data model. In addition, the relationships of abnormal objects discovered in diagnosis with respect to their adjacent objects are also kept in the system. Due to the relatively small number of abnormal objects found, we only need to maintain a small fraction of spatial relations in the system.

The number of spatial relations for n objects in an image is on the order of $O(n^2)$. Storing all the spatial relations not only increases the storage but also lengthens the time needed to search for the objects. To limit the size of storage of spatial features and relations, only those most frequently inquired about are stored in the system. There is a storage/response time tradeoff as to what spatial features and relations need to be stored.

Ontology of interested spatial relationships is provided to describe the topological and orthogonal relationships between a pair of objects (as shown in Figure 3.4). The set of important spatial relationship features provided by domain experts are used to define terms in the ontology. More important parameters for distinguishing the sub-types under a category are placed first in the list, and parameters used in the higher branches may also be used in their descendant branches. In Figure 3.4, `Bordering` means that only the surfaces of the two objects are joined (i.e., $r_c > 0, r_j = 0$); `Invading` implies their areas are joined (i.e., one of the object is deformed by the other, $0 < r_j < 100\%$), and `Circumjacent` implies a full containment (*i.e.*, one object fully contains the other, $r_j = 100\%$). The *required constructs/operators* are necessary for every spatial relationship. A *spatial relationship description table* (as shown in Table 3.2) lists the available parameters and semantic terms for describing the spatial relationships between two objects. The values of these spatial relationships are stored in the database.

This spatial modeling approach captures the containment and directional aspects of binary spatial relationships. Containment describes the relative position and possible *joining* relationship among objects while direction aspect describes the directional relationships from one contour to the other. The constructs for the spatial relationships are as follows:

1. No containment

   - `Disjoined`: This spatial construct is used to describe two objects that are separated from one another. (For example, the micro-lesion is disjoined from the lateral ventricle as shown in Figure 3.8). The

spatial relationship	**representative features**	**defined semantic terms**
SR(l,b)	$(x_c, y_c, r_a), (a_i)$	Slightly_Occupied, Extremely_Occupied
SR(l,f)	(x_c, y_c, r_j)	Slightly_Touching, Intimately_Touching
SR(l,lv)	$(\theta_c, d_c, x_c, y_c)$	Nearby, Far_Away
...	...	...

Table 3.2: A *spatial relationship description table* for the brain lesion (l: lesion; b: brain; f: frontal lobe; lv: lateral ventricle).

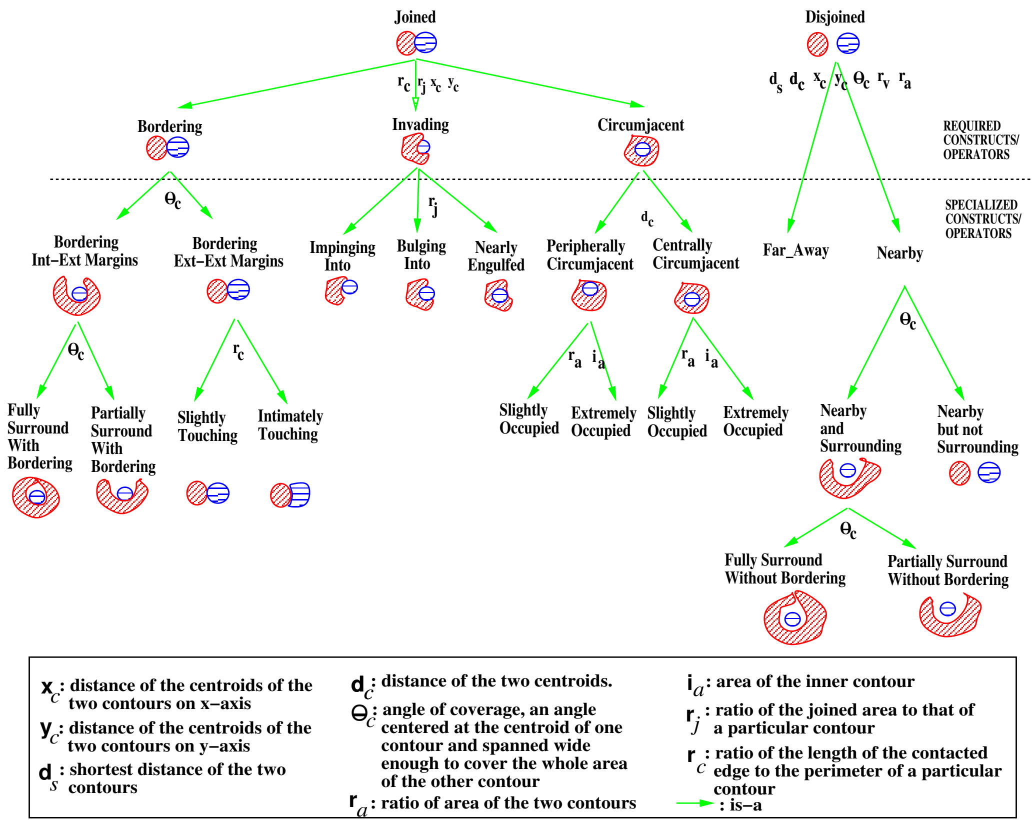


Figure 3.4: An ontology of spatial relationships (represented in a generalization hierarchy) for different topological categories between two objects (with the representative icons shown) for oncological medical images. The parameters under a branch classify the sub-types under that category.

relationships under this construct are further quantified by the spatial relationship features such as d_s, d_c, r_a, and θ_c.

The values of these spatial relationship features can be conceptually classified via clustering algorithms such as MDISC [CCH96] to derive the meaning of the corresponding semantic terms under the `DISJOINED` category, such as `NEARBY` and `FAR_AWAY`. The reason for using an instance-based clustering algorithm to define these corresponding semantic terms is that the semantics of these detailed spatial relationship terms are context-dependent (e.g., the meaning of `NEARBY` in astrology is different from its meaning in medicine).

- `Bordering`: Two objects contact one another at one or more locations. Bordering is a significant event in the medical domain. The relationships under this construct are further quantified by spatial relationship features such as r_c and θ_c.

2. Containment

   - `Circumjacent`: An object contains another object (for example, a micro-lesion develops inside the brain, as shown in Figure 3.8). The relationships under this construct are further quantified by the spatial relationship features such as d_c, r_a, and i_a.

3. Partial Containment

   - `Invading`: This spatial construct is used to describe one object that invades the other which causes an object to deform. The relationships under this construct can be further quantified by the spatial relationship features such as r_j and d_c.

The directional information of spatial relationships is captured by spatial relationship features such as the relative distance of the centroids of two objects on the x-coordinate (denoted as x_c), on the y-coordinate (denoted as y_c), and on the z-coordinate (denoted as z_c).

To model a spatial relationship, domain experts need to identify the containment categories of the modeled spatial relationship. Spatial relationship features are then identified to describe the spatial relationship according to the topological categories of the two objects.

For example, in an image with a lesion and lateral ventricle, the spatial relationship instance between the lesion and lateral ventricle is classified as an instance of the class SR(lesion,lateral_ventricle). The containment relationship covers the `DISJOINED` and `BORDERING` categories and requires θ_c, d_c, x_c, and y_c as the features to describe this spatial relationship. Figure 3.5 is an image classification hierarchy for the images in the database. The hierarchy is generated by MDISC based on the collected values of spatial relationship features of SR(lesion,lateral_ventricle) instances in the database.

3.2 Temporal Content Modeling

Based on the lifespan (i.e., valid time [Sno87]) and class that a temporal object belongs to, the temporal behaviors of objects over time can be queried. The queryable temporal behavior includes:

1. Temporal relationships among objects,
2. Evolutionary phenomenon of objects, and
3. Stream behavior of objects.

3.2.1 Temporal relationships

Possible temporal relationships among two objects are illustrated in Figure 3.6 [LG93]. These relationships are used to express temporal constraints of interested objects. Valid time of the objects are compared to answer queries with these

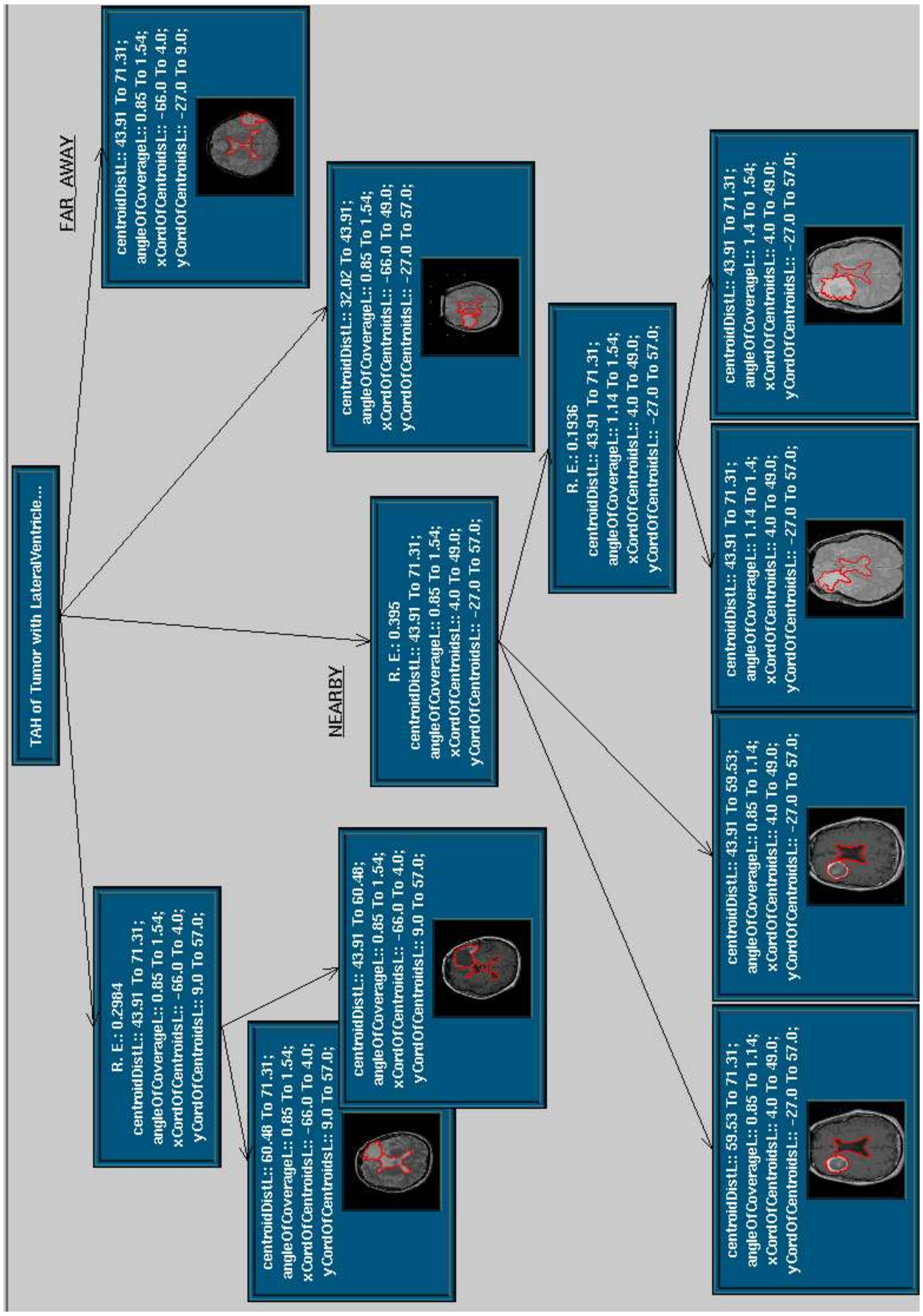


Figure 3.5: The MDISC-generated TAH (based on d_c, θ_c, x_c, and y_c) for representing the spatial relationship between tumor and lateral ventricle

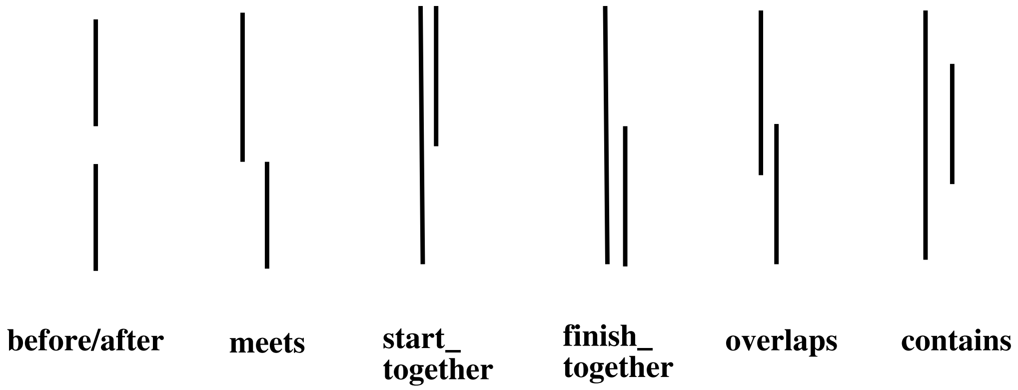


Figure 3.6: Temporal relationships of two temporal intervals

temporal constructs.

3.2.2 Evolutionary Object Constructs

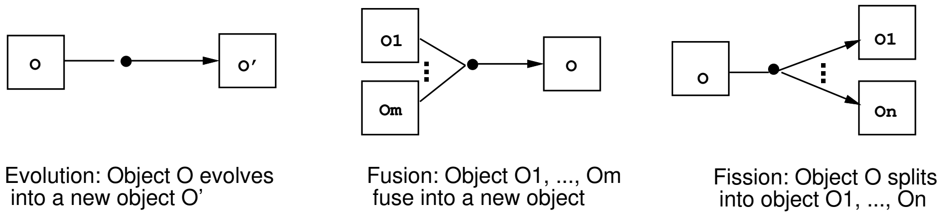


Figure 3.7: Evolutionary object constructs

Existing approaches for modeling object evolution involves maintaining object versions [KC88]. We enhance these basic models to describe the development of objects using *stages* and the *evolution*, *fission*, and *fusion* among stages. A stage represents a distinguishable development status of an object, and a transition between stages represents a significant change in the entropy or growth status of the evolutionary object. Evolution, fusion, and fission are the three possible types of transitions among the evolutionary stages (Figure 3.7) [CIT94].

1. Evolution: The characteristics of an object may evolve with time. For example, a micro-lesion evolves into a macro-lesion over a period of time, as shown in Figure 3.8.

2. Fusion: An object may fuse with other objects to form a new object with different characteristics than either of the constituent objects.

3. Fission: An object may split into two or more independent objects.

Evolution, fusion, and fission are used as query constraints to express evolutionary conditions of interested objects.

3.2.3 Stream Constructs

Time-related objects can be ordered and represented as a stream. The temporal and evolutionary predicates can be considered as constraints on the relationship of snapshots in streams. Thus, streams can be used as a queryable unit to integrate spatial, evolutionary, and conceptual constructs [DC96]. In this dissertation, we focus on the language extension and query processing for streams.

3.2.4 Example

The clinical manifestations of a lesion depend on its position with respect to its surrounding brain structures. As an example, a brain lesion is modeled with the temporal, evolutionary, and spatial object constructs as shown in Figure 3.8. Micro-lesions develop **separately** from (modeled in Figure 3.4) the lateral ventricle. Micro-lesions may also **evolve** (modeled in Figure 3.7) into macro-lesions which **invade** (modeled in Figure 3.4) the symmetrical lateral ventricle and make the lateral ventricle become asymmetrical.

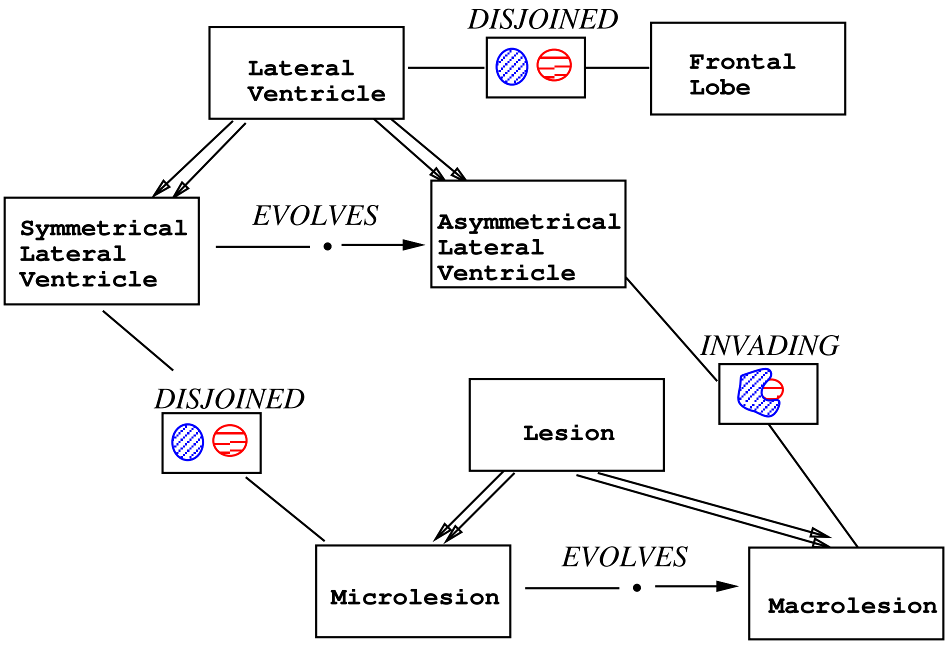


Figure 3.8: The temporal evolutionary and spatial relationships for the growth of a brain lesion.

CHAPTER 4

Intelligent Retrieval

Knowledge is required for intelligent retrieval. In this dissertation, we use clustering-based approach on (spatial) image features to provide the knowledge for answering conceptual terms and similar-to and semantic operators. Clustering-based approach is used since it provides the conceptual hierarchy that match users' perception. Individual nodes in the clustering hierarchy can also be labeled with conceptual terms (e.g., large, heavy) and/or semantic operators (e.g., nearby, far_away) to provide the use of these conceptual terms and semantic operators in query specification and processing. Moreover, the clusters can be automatically generated based on feature values of database instances to provide a scalable solution for intelligent retrieval.

4.1 Clustering-based Approach to Support Conceptual Constructs in Queries

The criteria of our image feature clustering algorithm is to minimize the averaged pair-wise Euclidean distance of image feature values in a cluster [CCH96]. Such a measure, known as the *relaxation error*, considers both the *frequency* of the value occurrence and the difference between values. Based on minimizing the summed *relaxation error* of all the newly partitioned clusters in each iteration, the clustering algorithm, MDISC, recursively partitions the data set to generate a multi-attribute clustering hierarchy, called multi-attribute type abstraction hierarchy (MTAH). Since both the feature value distribution and the correlation among different attributes of a feature are considered in MDISC, our clustering algorithm provides better image feature classification than if we used standard deviation to represent image similarity [BPJ93]. This automatic generation of knowledge based on the selected features of images makes our knowledge-based query answering approach scalable.

4.1.1 Query Interpretation via TAH

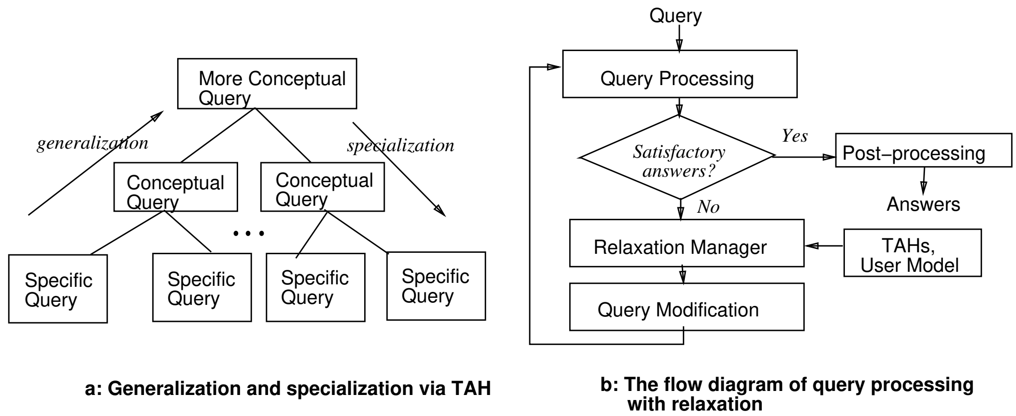


Figure 4.1: Knowledge-based query relaxation

The classification of image features are represented in a hierarchical representation, called *type abstraction hierarchies* [CC94, CYC96] for processing *similar-to* and semantic operators. The concept in the TAH nodes is represented by the feature value ranges (see Figures 3.2 and 3.5). These value ranges can be used to retrieve similar images. As shown in Figure 4.1.a, higher nodes in the TAH represent more generalized concepts (i.e., broader range of feature values) than that of the lower nodes (i.e., narrower range of the feature values). Based on the existing values of the representative spatial relationship features, MDISC can be used to classify image features to represent these semantic operators. The resultant TAH nodes are labeled by domain experts with the conceptual terms (e.g., `nearby`, `far_away`, `large`, `small`, `upper_protrusion_pressed_to_the_right`) to represent the specific domain concept of the images under the node. The value range is used as the query constraint to retrieve the images satisfying the conceptual predicates[1]. These conceptual terms are selected from the *domain dependent descriptive ontologies*. For example, Figure 3.4 is an ontology for various spatial relationships applicable to oncological medical images. A descriptive ontology for a specific image domain provides 1) the conceptual terms for labeling the TAH nodes, and 2) the shape, spatial relationship, and/or temporal features for clustering the object instances. These ontologies can be obtained from generic medical terminology (e.g., UMLS [US92]) and domain experts. They provide a restricted set of vocabularies for querying image content. Note that the proposed ontology for describing entities (e.g., size, color, distance) are different from the common sense generic ontologies such as Cyc and WordNet[Mil95] since our proposed ontology is more application-specific and the value ranges for each term in the ontology can be generated accordingly based on database instances.

To solve a *similar-to* query based on the spatial relationship features which has been classified and the classification information is stored in a TAH, the lower TAH nodes are attached with more specific value ranges. In solving the *similar-to* query, we shall first locate the TAH node that has a value range closest

[1] The conceptual terms can also be labeled via other approach (e.g., neural networks, vector computation, etc.). Our distribution-sensitive clustering algorithm, which is based on the selected features, provides sufficient classification information for the domain experts to label the TAH nodes.

to that of the target image. By traversing up (i.e., generalizing) and down (i.e., specializing) the selected TAH, the feature value range in the matched TAH node can be used for retrieving similar images from the database (Figure 4.1.b). The traversal along the TAH nodes is controlled by the relaxation policy which is provided either directly from users or from the user model.

There is a *TAH directory* in the system that stores such information as object names, sets of features, spatial relationships, user type of the TAH. Based on this information, the system (or user) selects and retrieves the appropriate TAHs for processing the query. If the retrieved TAH does not match the user's specification, the TAH can be edited to meet the user's requirement.

The time complexity to generate a multi-attribute hierarchy by MDISC is $O(mnlog(n))$, where m is the number of attributes and n is the number of distinct instances used in generating the TAH [CCH96]. Our experiment reveals that generating a MTAH with about one hundred images based on four features takes a fraction of a second's processing time on a Sun Sparc 20 workstation.

4.1.2 User Model

The user model specifies the user preference for customizing a query so that it is context and user sensitive. The preference of a user under a query context is characterized in a user profile.

In a user profile, the query context to be matched is represented by two different types of objects: *mandatorily matched objects* and *optionally matched objects*. *Mandatorily matched objects* of a user profile specify the objects that must be matched with a given query for the user profile to interpret the query. *Optionally matched objects* provide guidelines for adding additional object feature constraints when the corresponding objects are matched. Optional matching permits a partial matching of user profiles based on the objects appearing in the query and increases the matching occurrences.

The user preference specifies the set of interested features and spatial relationships for a given query context, and the policies for relaxing query conditions when no satisfactory answer is found.

The relaxation policy describes how to relax the selected TAHs when no satisfactory answer is found, since classifying images may involve multiple attributes as well as multiple TAHs to provide more efficient and accurate query answering. Control operators such as the *relaxation order* (e.g., which TAHs should be relaxed first), non-relaxable features, preference list, and relaxation level can be used to provide relaxation control.

For example, given MR brain images with lesion(s), the features that are of interest to a typical brain surgeon include the lesion locations and the spatial relationships with other objects in the brain (such as lateral ventricles) as shown in Figure 4.2. The information in the user model can be used to select the interested features and the relaxation policy if they are not specified in the query. For a given image, different types of users (e.g., radiologists, surgeons, and clinicians) may have different interests which can be specified in the user model.

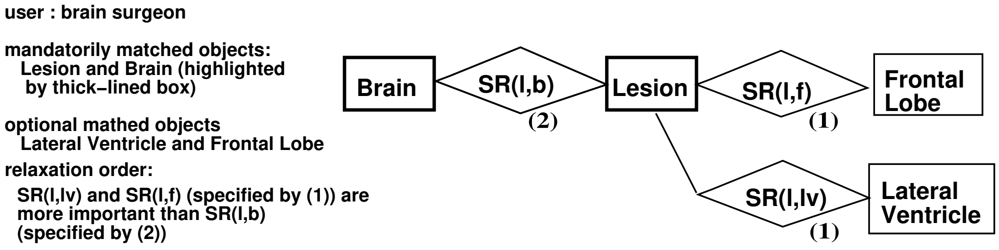


Figure 4.2: A user profile for brain surgeons

CHAPTER 5

Query Language

5.1 Introduction

In many image database applications, such as medicine and geographical information systems (GIS), image objects are spatial, temporal, and evolutionary. In feature-based querying, the precise specification of feature values are often very difficult if not impossible. Using semantic operators (such as *similar_to*, *near_by*), conceptual terms (such as *large*, *heavy*), and approximate operator mimics human cognitive process and expression, and thus greatly eases query constraint specification.

Query languages [OM88, HCT96] have been proposed to query image content. However, none of them support *conceptual, spatial, temporal, evolutionary, and stream querying* in an integrated query language. Therefore, in this chapter, we propose a Knowledge-based Spatial Temporal Query Language (KSTL) for image databases. To leverage on available commercial technology, constructs in KSTL are extended from ODMG's OQL [Car94].

The KSTL constructs are presented in Section 5.2. The methodology of extending KSTL constructs via ODMG's OQL is discussed in Section 5.3. Details of the extended KSTL predicates are presented in Section 5.4. Examples from medical images and GIS applications are used to illustrate the expressiveness of

KSTL.

5.2 Knowledge-Based Spatial Temporal Query Language (KSTL) Constructs

The following are the types of constructs proposed in our KSTL for supporting conceptual, spatial, temporal, evolutionary, and stream querying.

- *conceptual terms and approximate values for querying attribute values* (e.g., *large-sized* tumor, tumor size *about* 10 mm);
- *semantic terms for expressing spatial relationships of objects* (e.g., object A *nearby* object B, object A is *north to* object B)
- *object similarity*: searching objects similar to a target object based on a set of specific features.
- *temporal relationship constructs of objects* (e.g., *before* and *after*).
- *evolutionary behavior of objects*: *Evolve*, *fusion*, and *split* [CIT94]. Evolutionary predicates allow users to express constraints of specific object evolutionary trails.
- *stream behavior*: A stream [DC96] is a sequence of objects. A stream can be queried based on either the feature values of individual snapshots in the stream or the trend of feature value changes in the stream.

5.3 Extension of KSTL Constructs via ODMG's OQL

The KSTL constructs can be extended from OQL with minor syntax modification. Commercially available OQL query processors can thus be used as the back-end engine to retrieve data. ODMG's OQL can be extended based on the following three approaches:

Methods : A method adds querying capability to a class in object-oriented data model (e.g. $student.averageGPA()$). Methods of a superclass can be used by all the instances inheriting the superclass. The type system handles the polymorphism of method invocation. Frequently used spatial, temporal, evolutionary, and stream querying constructs can be provided as *methods* of appropriate superclasses, from which other database instances can inherit. Thus, the OQL query processor can directly process these extended querying methods without any modification.

Functions : Frequently used functions can be implemented as functions. However, since the polymorphism of functions (i.e., multiple functions with the same name but expecting input parameters of different classes) needs to be handled explicitly by programmers (e.g., using CASE statements), it is not a good approach for language extension in OQL unless no polymorphism is expected.

Operators : In ODMG's OQL, operators can be query commands (e.g., SELECT-FROM-WHERE) or mathematical operators (e.g., $+, - =, <>$). Operators are also used to incorporate SQL's aggregate operators into OQL [Car94]. Adding any additional operator results in a change in syntax and requires additional query pre-processing. Thus, operators should be the last resort for OQL extension.

Since polymorphism is handled automatically by the type system in method invocation, the use of methods is preferred in extending OQL. Constructs whose inputs are always of the same class can be extended using functions. Operators are used only when methods and functions cannot satisfy the requirements of the extended query constructs. Similar to regular OQL predicates, the extended predicates can be combined with other OQL predicates by using logical connectors such as AND and/or OR.

5.4 Extending KSTL predicates via ODMG's OQL

In this section, we shall discuss specific KSTL predicates based on its extension from ODMG's OQL.

5.4.1 Conceptual Terms and Value Approximation for Querying Attribute Values

Conceptual terms and approximate values are useful constructs in querying image features since precise feature value range are difficult to specify. We use methods **IS** and **approximate** to extend OQL for querying attribute values using conceptual terms and approximate values. Based on OQL's method invocation, the grammar of this extension expressed in BNF [Car94] is:

predicate_with_conceptual_term::=`object.feature.`**IS**(`conceptual_term`)

approximate_predicate::=`object.feature.`**approximate**(`feature_value`)

Users specify the object, features, conceptual terms, and approximated values of their query in `object`, `feature`, `conceptual_term`, and `feature_value`. Any conceptual terms and approximated values can be expressed in the querying since they are provided as parameters inside the **IS** and **approximate** methods.

Query 1: "Find all 'large'-sized tumors from the tumor image databases and the patient is approximately 35 years old." The corresponding KSTL is:

```
FROM  Patients p
WHERE p.tumor.size.IS(``large'') AND p.age.approximate(35)
```

To process the query, the conceptual terms are translated into specific value ranges derived from the knowledge base [CYC96, HCT96]. The set of available conceptual terms for a specific attribute is stored in the system. The approximated value for an attribute is transformed into value range(s) from the knowledge bases [CYC96].

5.4.2 Semantic Terms for Expressing Object Spatial Relationships

The ontology of spatial relationships among objects are context- and application-dependent. For example, the directional information in GIS applications divides spatial relationships into *north of, south of*, etc., while based on the distance between objects, spatial relationships are divided into *far_away*, *near_by*, etc. In medical applications, a growing object's effects on others is often the primary concern. Based on distance, ratio of contact, and relative size of two objects, the spatial relationships can be divided into *far_away*, *bordering*, *touching*, *invading*, etc. Thus, for an application domain, the ontologies may be specified by the domain experts based on geometry (e.g., overlapping), domain information, or the clustering information derived from database instances [CCH96, CYC96, HCT96]. The labels inside the ontologies (such as *far_away* and *near_by*) are useful semantic terms that can be used to query the spatial relationships.

KSTL incorporates semantic terms to query spatial relationships without changing the syntax of the query language. Every object whose spatial relationship with others can be queried via semantic spatial relationship terms are called *spatial objects*. The method **srWith** combines two compared spatial objects into a relationship. The relationship is modeled as a class which can compute the features of the relationship (e.g., the shortest distance between two objects). This returned relationship is then queried with semantic terms using the **IS** method. Based on OQL's method invocation, the grammar of this extension expressed in BNF is:

sr_semantic_predicate::=`object`.**srWith**(`object`).**IS**(`semantic_term`)

Users specify the objects and semantic spatial relationship terms of the query in `object` and `semantic_term`.

Query 2: "Retrieve all images with tumors *nearby* frontal lobe and *invading* lateral ventricle". The corresponding KSTL is:

```
SELECT t.image
FROM  Tumor t, Lateral_Ventricles l, Frontal_Lobes f
```

```
WHERE t.srWith(l).IS(``invading'') AND t.srWith(f).IS(``nearby'')
```

The process of generating definitions for each term using knowledge discovery algorithms is detailed in Chapter 3.

5.4.3 Object Similarity

The predicate for feature-based similarity retrieves objects similar to a target object. The similarity is computed based on the values of a set of specified features. The following is the BNF expression for our proposed similar to operator:

`similar_to_predicate`::=`object` **similar_to**(`object`) [`subQuery`] **based_on** `features`

`features`::=`feature`|`feature[,features]`

Users specify the objects and the set of features for similarity retrieval in `object` and `features`. The target object is retrieved based on the constraint expressed in the `subQuery`. Since a sub-query expression is required to specify the target object, the similar-to predicate can not be extended as a OQL method. The similar-to operator is thus added as an operator to OQL.

Since the *similar_to* operator modifies the BNF expression of ODMG's OQL, a pre-processor is needed to convert the similar_to operator into other predicates acceptable to the OQL processors.

Query 3: "Find images in the database that have similar spatial characteristics as the given image on the screen." The corresponding KSTL is:

```
select   p1, p1.image
from     Patients p1, pt
where    p1.image similar_to pt.image
                   [pt.image.SELECTED_ON_THE_SCREEN]
                  based_on t2.size, t2.location
```

SELECTED_ON_THE_SCREEN is a method return a particular selected image on screen. Based on the knowledge base (as well as the user type and target attribute values), the similar-to operator is translated into specific value ranges of the compared features [CYC96, HCT96]. When the based_on subclause is not specified, the knowledge needs to figure out the set of specific features to be used based on the user type and the corresponding user model (see Chapter 4.1.2). Such an example is also illustrated in Chapter 6.3 to demonstrate how the knowledge in the knowledge layer is used to resolve similar-to operator.

5.4.4 Temporal Relationship of Objects

In querying temporal relationships of objects, the query language needs to provide constructs to compare the relationships between two selected temporal entities. The required temporal operators are provided as methods of temporal objects (e.g., `object1.`**before**`(object2)`) which yields true or false based on the temporal relationship of the compared objects. Since both points and intervals can be compared in the temporal predicates, a new class `INTERVAL` is introduced. OQL uses class name as the constructor to create a new instance of the class. Thus, the syntax for creating an interval is:

`interval`::=**INTERVAL**[`object`, `object`]

Both intervals and temporal objects uses *methods* to support this required temporal relationship comparison. Based on OQL's method invocation, the grammar of this extension expressed in BNF is:

temporal_expression::=`temporal_entity`.`temporal_op`(`temporal_entity`)

`temporal_op`::=**after**|**before**|**contains**|**overlaps**|**start_together**|**finish_together**

`temporal_entity`::= `object`|`interval`|`temporal_constant`

`Temporal_constant` could be a specific date (e.g. "1/1/1990"), time (e.g. "01:30:32 1/1/1996"), or the reserved word `NOW` which refers to the current time.

Research effort has been focused on comparing the temporal relationships of objects [Sno87, LG93]. **before**, **after**, **contains**, **start_together**, **end_together**, **meets**, and **overlaps** are the required methods for temporal objects and intervals to compare their temporal relationships [LG93].

The ability to query attributes via approximate values in KSTL can be combined with the **INTERVAL** construct to query the elapsed time between two specified objects using approximate time duration. The method **length** is applied to instances of intervals which returns the interval length. Approximate querying is then applied to the `length` attribute. The specific value range of the interval length to be relaxed is derived from the knowledge base based on the object class and query context. **Query 4** provides an example of such query.

Query 4: "Find patients whose tumors could have been consistently growing for approximate two years already." The corresponding KSTL is:

```
SELECT  p
FROM    Tumors t1 t2, Patients p
WHERE   t1.before(t2) AND t1.size < t2.size AND
        INTERVAL[t1,t2].length.approximate(``2 years'') AND
        t1.patientName = t2.patientName
```

5.4.5 Evolutionary Behaviors of Objects

For objects which evolve from one class/status to another, evolutionary querying retrieves objects of specific evolutionary development. The possible class changes for an object in its lifespan include 1) changing to another class, 2) merging with other object(s) into a new class, 3) splitting into objects of (different) classes. `EVOLVE_TO`, `FUSED_FROM`, and `SPLIT_INTO` [CIT94] are the fundamental evolutionary querying constructs which represent these possible evolutionary steps. In KSTL, `EVOLVE_TO`, `FUSED_FROM`, and `SPLIT_INTO` are provided as methods of objects to support evolutionary querying. Based on OQL's method invocation, the grammar of this extension expressed in BNF is:

evol_predicate::=`object`.`evol_op`(`objects`)

`evol_op`::=**evolve_to**| **fused_from**|**split_into**

`objects`::=`object`|`object`[,`objects`]

Viewing the object's evolutionary path as a *directed* graph, we know topologically that the three shapes in Figure 5.1 are the only three possible node-to-node transitions if the number of fan-outs and fan-ins are neglected. Thus, any possible evolutionary changes can be expressed based on these evolution, fusion, and

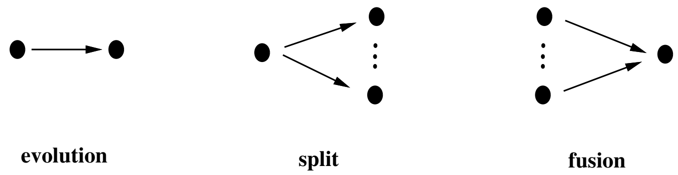


Figure 5.1: The possible node-to-node transitions of a graph

fission constructs.

Query 5: "Retrieving tumor instances which split on their evolution from stage_1 to stage_3 into two tumors of different sub-types stage_3A and stage_3B". The corresponding KSTL is:

```
SELECT  t1, t2, t3
FROM    STAGE_1_Tumors t1, STAGE_3A_Tumors t2, STAGE_3B_Tumors t3
WHERE   t1.split_into(t2, t3)
```

5.4.6 Stream Behavior

A stream [DC96] is a sequence of objects. Objects in a stream are called *snapshot (objects)*. A stream can be queried based on the feature values of individual snapshot objects or on the trend of feature value changes of consecutive snapshots. Most temporal database research [Sno87] has focused on the relationships among snapshots that satisfy a given set of predicates. However, constructs are needed to query streams themselves. For example, in Query 4, by checking the tumor size change between the two referenced snapshots, we only know that the tumor is growing but cannot conclude that if the tumor has consistently increased in size during the two years. Additional constraints should be specified on snapshots existing in this two year interval. Snapshots in this two year interval could be organized as a stream and the consistently increasing of the tumor size is a constraint to be applied on the stream. Our extension allows streams and their substreams to be referenced in the query to specify constraints on all the contained snapshots.

In querying the characteristics of streams, the predicates can be *snapshot predicates* and *stream predicates*. A snapshot predicate has snapshot object(s) as its operand and returns *true* if the snapshot satisfies the predicate. A stream predicate has stream(s) as its operand(s) and returns *true* if the trend of feature value changes within snapshots in the stream satisfies the predicate. Temporal logic is fully expressive of queries in historical data model [GM91, Gab89]. By showing that every temporal logic expression has its corresponding expression in KSTL, KSTL can provide the capability to express all temporal logic expressions.

To enhance the ability to express complex temporal constraints, temporal logic introduces two modal operators $\Diamond p$ and $\Box p$ to add to a snapshot predicate p [GM91, Gab89]. $\Diamond p$ is a new predicate which evaluates p within a stream s. $\Diamond p$ returns true if the current snapshot in s or a snapshot after the current snapshot satisfies p. $\Box p$ is another new predicate which returns true if all the snapshots after the current one (including the current snapshot) in s satisfy p. Complex temporal constraints can be expressed by combining different snapshot predicates and their $\Diamond$ and $\Box$ variations. Predicate $\Diamond p$ and $\Box p$ not only evaluate p on the current snapshot but also on other snapshots in the stream as well. Combining predicates $p_1 p_2$ (where p_1 and $_2$ may contain modal operator $\Diamond$ or $\Box$) implies that a particular snapshot exists in the evaluated stream where *all* the snapshots between this particular snapshot and the last snapshot in the evaluated stream satisfy p. We call p, $\Diamond p$, and $\Box p$ as *simple temporal logic expressions* and expressions composed from simple temporal logic expressions as *complex temporal logic expressions.*

For example,

always $p == \Box p$ (which implies that p hold all the time)

p until $q == \Box pq$ (which implies that p hold all time until the time q holds)

p since $q == (\Diamond - q)q(\Box p)$ (which implies that at the beginning q is false, and right after q turn true p holds all the time)

Directly using temporal logic to express database constraints for temporal data model has the following problems:

- *unclear variable binding.* For example, while evaluating the temporal logic expression $\Box pq$, the beginning and ending snapshots satisfying $\Box p$ are not bound to any variable.

- *unable to specify the interrelationship of snapshots occurring in different predicates.* Since no variable(s) are bound to the operand(s) of a temporal logic predicate, the interrelationship for snapshots appearing in different predicates cannot be specified. For example, in the expression $\Box pq$ an additional constraint cannot be added to restrict that the beginning snapshot satisfying $\Box p$ should be at least a specified time interval away from the snapshot satisfying q.

Current research on incorporating temporal logic into database query languages [Ric92, GM91] are mostly based on SQL which cannot provide as rich extension on data types and variable binding as that of ODMG's OQL. Thus, these existing languages suffer from unclear variable binding. KSTL provides clear variable binding and can express all temporal logic expressions.

Let $p(x)$ be a snapshot predicate expressed in KSTL. $p(x)$ corresponds to the predicate p in previous discussion and $\Diamond p$ and $\Box p$ are its two variations in temporal logic. S is a set of streams to be evaluated against temporal logic expressions to retrieve streams satisfying the constraints. Note the following existing OQL commands are useful for stream querying:

- to evaluate $\Diamond p$ on a set of stream S, we can expressed it in OQL as

  FROM S as s, x in s WHERE $p(x)$

  Note that S is a set of stream. Thus s is bound to a stream in each iteration. Since the stream referred by s is also a collection of objects, it can appear in the FROM clause to have x bound with a member object in s.

- the `forall` expression in OQL can be used to express the expression $\Box p$ as

  FROM S as s WHERE `forall` x in s: $p(x)$

  `forall` is the expression supported in OQL for a collection.

- for any ordered collection s, OQL supports functions $first(s)$ and $last(s)$ to return the first and last object in the ordered collection.

In addition to these existing constructs in OQL, we need two additional functions as our extension on stream querying:

- method *next* is applied to any snapshot object x where $x.next$ return the next object of x in the same stream.

- for a stream s, we construct a new function $AllSubStream(s)$ which returns a collection containing all the substreams of the stream s.

These two extensions support complex temporal logic expression. $x.next$ lets us express the concatenation of two neighboring simple temporal logic expressions. Function $AllSubStream(s)$ generates a set of all the substreams in S that allows us to bind any candidate substream as variable(s) in order to evaluate the bound substreams in the WHERE clause.

With the above extensions and functions/methods $first$, $last$, $next$, $forall$ in OQL, we can translate all the temporal logic expression into our OQL extensions as follows[1]:

$\Diamond p ==$ FROM S as s, x in s WHERE $p(x)$

$\Box p ==$ FROM S as s

WHERE forall x in s: $p(x)$

$\Diamond p \Diamond q ==$ FROM S as s, x in s, y in S

WHERE $p(x)$, $q(y)$, $x.before(y)$

$\Diamond p \Box q ==$ FROM S as s, x in s, $AllSubStream(s)$ as z

WHERE $p(x)$, forall y in z: $q(y)$, x.next == first(z)

$\Box p \Diamond q ==$ FROM S as s, x in s, $AllSubStream(s)$ as z

[1] The complete translation algorithm is provided in Appendix B

WHERE forall y in z: $p(y)$, first(z) == first(s), p(x), z.before(x)

$\Box p \Box q$ == FROM S as s, $AllSubStream(s)$ as $z1, z2$

WHERE forall $y1$ in z: $p(y1)$, forall $y2$ in z: $q(y2)$, $first(z1) == first(s)$,

$last(z2) == last(s)$, $last(z1).next == first(z2)$

Thus, extending from OQL, KSTL is sufficient to express complex temporal constraints based on temporal logic. Clearer variable binding is provided in KSTL to express temporal constraints. Users may either compose the temporal logic expression first and then write it into corresponding KSTL, or directly compose the constraints in the KSTL.

Query 6: "Retrieve streams that has four specific snapshots and period between the snapshots as shown in Figure 5.2" The corresponding KSTL is:

```
SELECT  p, x1, x2, x3, x4
FROM    Patients p, p.tumorSequence as s,
           x1 in s, x2 in s, x3 in s, x4 in s,
           TumorImages t1, t2, t3, t4
WHERE   p.treatment.before(x1) AND p.race=='African American' AND
           x1 similar_to t1 [t1.imageID='001']
              based_on t1.tumor.size, t1.tumor.location AND
           x2 similar_to t2 [t2.imageID='002']
              based_on T1.tumor.size, t1.tumor.location AND
           x3 similar_to t3 [t3.imageID='003']
```

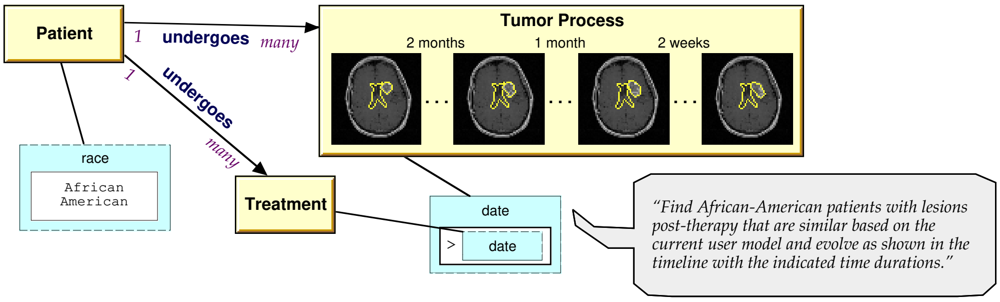


Figure 5.2: A visual query expression for **Query 6**

```
      based_on t1.tumor.size, t1.tumor.location AND
  x3 similar_to t4 [t4.imageID='004']
      based_on t1.tumor.size, t1.tumor.location AND
  INTERVAL[x1,x2].approximate(''2 months'') AND
  INTERVAL[x2,x3].approximate(''1 month'') AND
  INTERVAL[x3,x4].approximate(''2 weeks'')
```

Here, the method `tumorSequence` returns only one tumor sequence for each patient. If the method `tumorSequence` can return more than one sequence, additional variables should be used to ensure that the variables `x1, x2, x3, and x4` refer to the same single stream in the query. Further query optimization are left to the OQL query optimizer.

Moreover, the predicate p (which is limited as a snapshot predicate in the discussion so far) can be generalized to a stream predicate. For example, let t be a tumor size stream. $p'(t)$ is a stream predicate which returns true if the size changing pattern of t is *similar to* the pattern of a particular given target stream. Then $\Diamond p'$ should be a valid temporal constraint to retrieve substream(s) *anywhere* inside the data streams whose size changing pattern is similar to that of the target stream. On the other hand, since the semantics of $\Box p'(t)$ is unclear, this generalization on adding modal operator to stream predicate is thus limited to adding only the modal operator $\Diamond$ and not $\Box$. The function $AllSubStreams(S)$ in KSTL returns a set containing all the candidate substreams. This returned set is used in the FROM clause of a KSTL statement for variable binding. The bound variable is used in the WHERE clause to express the constraints implied in $p'(t)$. Thus this generalization is expressible in KSTL. Query 7 illustrates an example of this extension.

Query 7: "Retrieve streams containing sub-stream(s) similar to the size change pattern of John Doe's tumor size stream." The corresponding KSTL is:

```
FROM    Persons  p1 p2, p1.tumorSizeSequence  s,
        AllSubStream(s)  z
WHERE   z similar_to p2.tumorSizeSequence [p2.patientName='John Doe']
               based_on p2.tumorSizeSequence.size
```

Query 8: "Find all the thermal emission which have been inactive for a long time and become active recently." The corresponding KSTL is:

```
SELECT last(vz)
FROM VehicleSequenceCollection vs, AllSubStream(vs) vz
WHERE forall x in vz: x.thermalEmission.IS(``inactive'') AND
      last(vz).next.thermalEmission.IS(``active'') AND
      vz.length.IS(``long_time'') AND
      last(vz).next.time.approximate(NOW)
```

The sequence of a vehicle's movement is stored as a stream and all streams of moving vehicles are stored in the set `VehicleSequenceCollection`. `length` is the method to extract the time period of a stream and NOW is a reserved word which refers to the the current time. `long_time` is a conceptual term to query the length of the interval and it is converted into specific range of time length derived from the knowledge-base.

Query 9 "Find all cases in which a tumor decreased in size for less than three months post treatment, then resumed a growth pattern after that period." The corresponding KSTL for Query 9 is:

```
SELECT p, s
FROM   Patient p, p.tumorStream s, AllSubStream(s) z1, z2
WHERE  forall x1 in z1: x1.sizechange < 0  AND
       forall x2 in z2: x2.sizechange > 0  AND
       z1.length < ''3 month'' AND
       last(z1).next = first(z2) AND
       p.thermalAblation.before(z1)
```

`last` abd `first` are functions in OQL which retrieve the first and last element in an ordered collection. `next` is a method in KSTL which retrieves the immediate downstream snapshot of the current snapshot. Thermal ablation is one of the therapy for treating tumors and the method `thermalAblation` returns patient's record for this treatment. A visual query expression using MQuery[DC96] is shown in Figure 5.3 to visually illustrate the query.

Predicates can be used to express constraints on sub-stream as long as the corresponding methods are available to evaluate against streams. Since not all the sub-streams in the *AllSubStream(aStream)* need to be examined to evaluate the query, a query processor should optimize the evaluation of the function

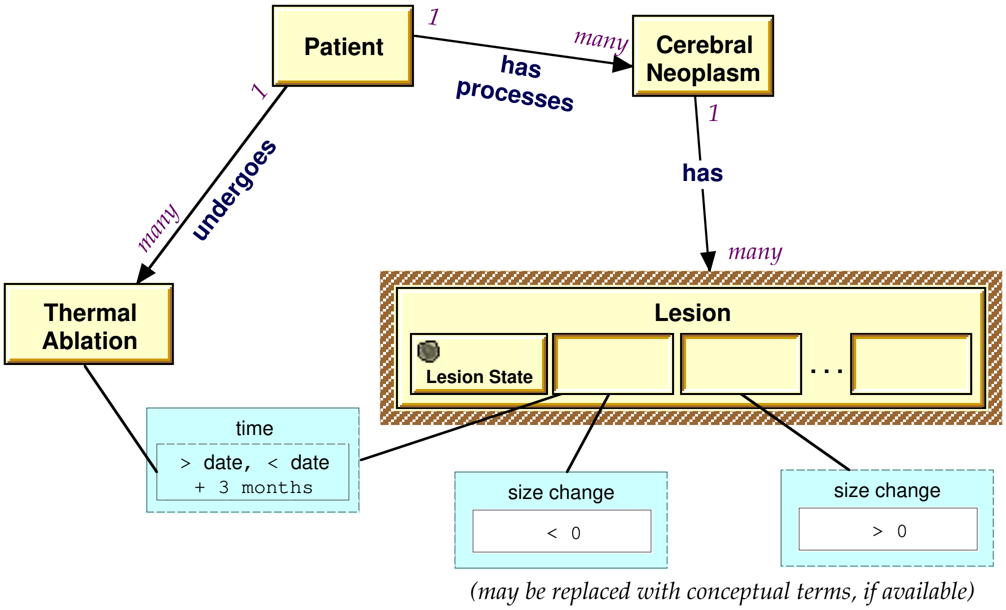


Figure 5.3: Visual query expression for Query 9

$AllSubStream(aStream)$. This optimization can be accomplished using a filtering process such as the optimization technique used in string and list retrieval [SLV95]. For example, if the beginning or ending snapshot is required to satisfy certain constraints, only those sub-streams that begin or end with snapshot satisfying the constraints are examined. Due to its high execution cost, examining all candidate substreams in the *AllSubStream(aStream)* set should always be a last resort for processing the stream predicate.

Query 10: "Retrieve the image frames in which a micro-lesion is nearby the lateral ventricle and approximately 9 mm in diameter. The micro-lesion evolves into a macro-lesion with diameter equal or larger than 25 mm and invades the lateral ventricle in approximate one year". The corresponding KSTL is:

```
SELECT  mi.image, ma.image
```

```
FROM    MicroLesions mi, MacroLesions ma, LateralVentricles lv1, lv2
WHERE   Interval(mi,ma).length.approximate(``1 year'') AND
        mi.evolve_to(ma) AND lv1.srWith(mi).IS(''nearby'') AND
        mi.diameter.approximate(9) AND
        ma.srWith(lv2).IS(''Invades'') AND
        ma.diameter >= 25mm AND lv1.patient = lv2.patient
```

Queries 7, 8,9 and 10 show the expressive power of KSTL in specifying the conceptual, spatial, temporal, and evolutionary image object content in a query.

CHAPTER 6

Query Processing

During the query processing, the proposed conceptual constructs and extended functions/methods for temporal, evolutionary, and stream behaviors should be converted into commands that can be processed by commercially available OQL query processor [Car94]. Table 6.1 categorizes the extended OQL constructs to be implemented for KSTL.

Data in the image databases are organized into class hierarchies based on their data structures and required computational and querying functionalities. Common property such as spatial, temporal, evolutionary, and stream property can be implemented in the top level superclasses of the class hierarchies. These top level superclasses are called *kernel-classes*. We use a kernel class-based approach to implement the extended functions/methods. Thus, the commercially available OQL query processor [Car94] can invoke the methods and functions in the implemented kernel classes to directly process the KSTL query. On the other hand, the conceptual constructs in KSTL need to be converted into commands that OQL query processor can directly process. A *knowledge-based query modification technique* is used [CYC96, HCT96] where the conceptual constructs in KSTL are converted into specific value ranges of image features based on the related knowledge in the knowledge layer.

Based on the kernel class-based technique and knowledge-based query modi-

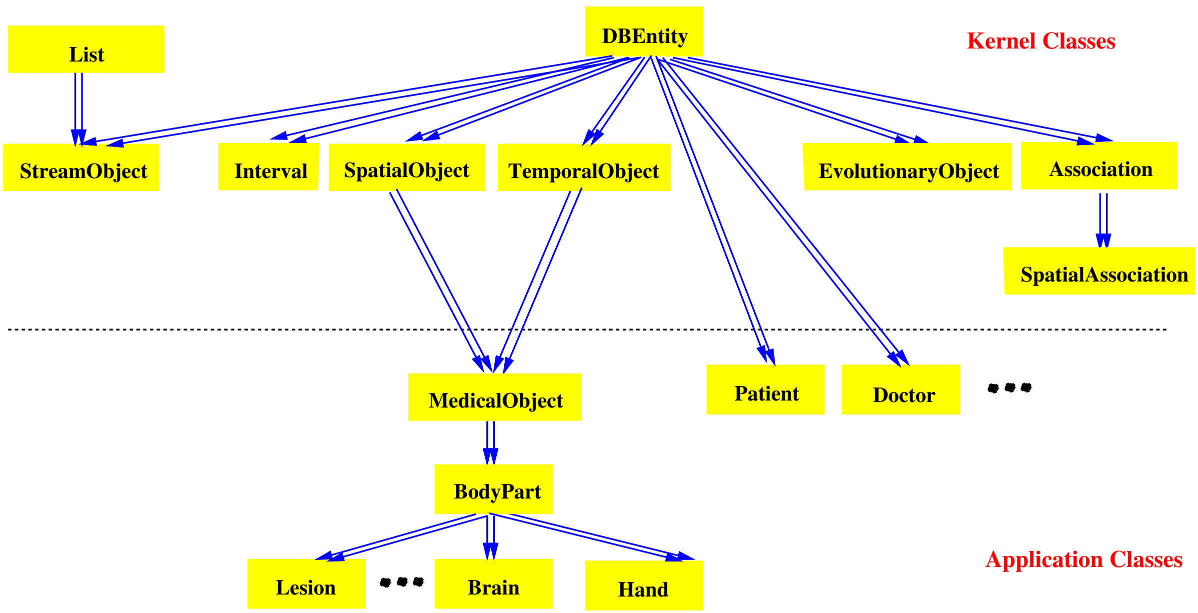


Figure 6.1: The kernel classes implemented in the KMeD system

fication, the flow of processing the KSTL queries is presented in Section 6.3.

6.1 Kernel-Class-Based Implementation of Spatial, Temporal, Evolutionary, and Stream Constructs

The kernel classes are constructed as shown in Figure 6.1. `DBEntity` implements the fundamental functions required for all objects in the image database. Generic methods for spatial objects are implemented in the *SpatialObject* class (e.g. the **srWith** method). Generic methods for temporal objects are implemented in the *TemporalObject* and *Interval* class (e.g., `before` and `after`). The extended methods for evolutionary and stream objects are implemented in the *EvolutionaryObject* and *StreamObject*. Details of the kernel classes are described as follows.

	required OQL methods	*required OQL operators*	*required OQL function*
conceptual query answering	IS	similar_to based_on	
spatial querying	srWith, IS		
temporal querying	before, after, meets, contains, overlaps, start_together, finish_together		
evolutionary query	evolvo_to, fused_from, split_into		
stream querying	next	in, similar_to based_on	AllSubStream

Table 6.1: Categories of required methods and operators in the proposed OQL extension

6.1.1 SpatialObject Class

All objects which occupy a position in space are instances of the SpatialObject. Common representations for a spatial object (such as raw image(s) and the *contour* which represents the boundary of the object) and algorithm for computing features of the spatial objects (e.g., *area* and *perimeter* of the object computed based on the contour) are implemented in the SpatialObject class. Values of these computed features are used to answer spatial queries. The method **srWith** combines a pair of compared spatial objects into a relationship. The relationship is modeled as the `SpatialAssociation` class which can compute the features of the spatial relationship (e.g., the shortest distance between two objects). This returned relationship is then queried with semantic terms using the **IS** method.

6.1.2 SpatialAssociation Class

The required computation for comparing spatial relationships among two objects is implemented in *SpatialAssociation*[1]. *SpatialAssociation* class contains features that describe the spatial relationships of the two objects (e.g., the nearest distance of the two contours, angle of coverage [HCT96], etc.) The computation algorithms for these features are implemented as methods in the *SpatialAssociation* class. Sub-classing under SpatialAssociation is allowed for different application domains. Since the *SpatialAssociation* class stores values of interested spatial relationship features, semantic terms can be evaluated directly on instances of *SpatialAssociation* class.

[1] Similar approach can be extended to model spatial relationships of more than two objects.

6.1.3 TemporalObject Class and Interval Class

Objects whose lifespan (i.e., valid time [Sno87]) is recorded in the database are instances of *TemporalObject*. The required temporal relationship comparison (as described in Section 2.3) are implemented as methods of *TemporalObject*. TemporalObject supports the method, *time*, which retrieves the recorded valid time. Using the method *time* to extract the valid time of an object's lifespan, the methods that compare temporal relationship (e.g., **before** and **after**) are implemented by comparing the extracted time. For example, *object1*.**before**(*object2*) in KSTL is implemented as *object1*.time < *object2*.time. Other temporal comparisons are implemented similarly.

The class `Interval` in KSTL takes two temporal objects as inputs and returns a time interval which spans over the lifetime of the two objects (including the lifetime of both objects). The required temporal relationship comparison operations for intervals are thus implemented as methods which compare the time extracted from the intervals.

6.1.4 EvolutionaryObject Class

An evolutionary object is an object which may change its object class during its lifespan. *EvolutionaryObject* is the kernel class which implements the common data structure and required functionality for evolutionary objects. Information required to track object's evolutionary behavior is represented and stored in the *EvolutionaryObject* class.

Several versions of the same object may exist in the database (e.g., more than one MR image which shows the same brain tumor in a patient's brain are different versions of the tumor existing over time). This evolutionary phenomenon is further enhanced with evolutionary constructs (i.e., `evolvo_to`, `split_into`, and `fused_from` [CIT94]) in KSTL. Thus, *EvolutionaryObject* needs to explicitly track this evolutionary development. In the evolution of objects, one object may evolve into another object of different status, split into many different objects; or multiple objects may merge into one object. In the implementation, a *system generated evolutionary sequence ID* (ESID) is assigned for each *simple evolutionary sequence* during the data insertion time, and the ESID is stored in every instances of the sequence (see Figure 6.3(a)). A *simple evolutionary sequence* is the evolution of an object without any fission or fusion involved. A binary table (called *evolutionary tracking table*) is used to track the split and fusion among evolutionary sequences as shown in Figure 6.3(d). The evolutionary query constructs (i.e., `evolvo_to`, `split_into`, and `fused_from`) are implemented as methods which determine the evolutionary information based on these tracked evolutionary information. For example, evolutionary predicate `object1`.**evolve_to**(`object2`) is implemented as `object1`.ESID == `object2`.ESID where ESID is the method to retrieve the ESID of an EvolutionaryObject instance. Evolutionary predicate `object1`.**fused_from**(`object2`, `object3`) is implemented as checking whether the two tuples (`object1`.ESID, `object2`.ESID) and (`object1`.ESID, `object3`.ESID) are both in the evolutionary tracking table in Figure 6.3(d). The evolutionary tracking table can be hashed based on ESID to improve the performance. This ensures the execution efficiency of the proposed evolutionary query constructs.

Figure 6.2 shows the retrieved image sequences from *Query 10* using the implemented evolutionary methods.

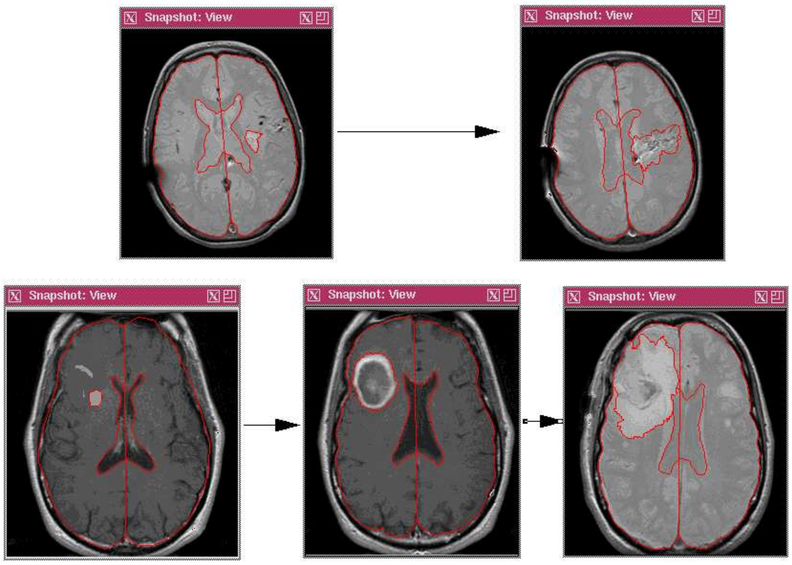


Figure 6.2: The returned image sequences showing the macro-lesion invading the lateral ventricle for the *Query 10*

6.1.5 StreamObject Class

In extending OQL to allow stream as a queryable unit based on temporal logic [HC97], the methods `in,` `next`, and `previous` and function *AllSubStream(aStream)* need to be implemented. Lists are required as the physical schema to store the snapshots in a stream since the total ordering of snapshots in a stream is required in the physical storage level to support these required extended methods. For example, finding a snapshot *next* to the currently referenced snapshot is a simple operation when the snapshots in a stream is physically stored in a list but a complex operation if the snapshots in a stream are stored in a non-ordered set.

Since OQL supports lists and the extended methods `in,` `next`, and `previous` are supported methods in lists, these extended methods/operators for streams can be directly passed to the OQL query processors for data retrieval. The function *AllSubStream*(*aStream*) is implemented to return a set of lists where each list

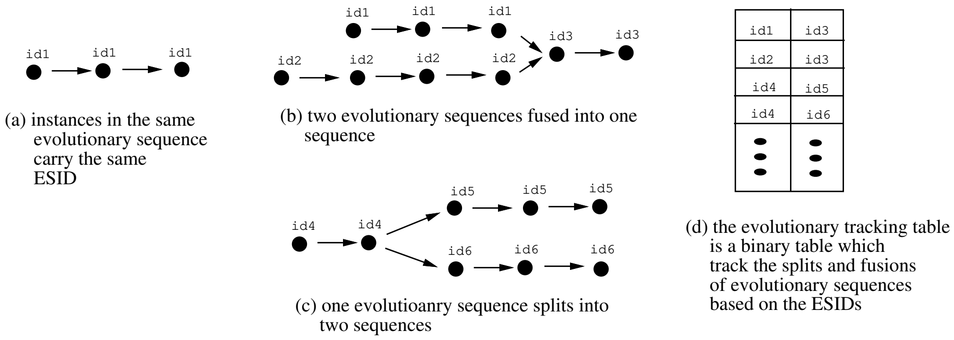


Figure 6.3: Tracking the evolutionary behavior of objects by the Evolutionary-Object class

represents a substream of $aStream$.

The kernel class *StreamObject* is a subclass of list. *StreamObject* stores all the contained snapshots in a list. Stream instances of the same type are stored in a set (e.g., `VehicleSequenceCollection` in Query 8) and the set is used in query specification and processing to select satisfactory streams from it.

Since not all the sub-streams in the *AllSubStream(aStream)* need to be examined to evaluate the query, further optimization can be performed to evaluate the function *AllSubStream*(*aStream*). This optimization can be accomplished using a filtering process such as the optimization technique used in string and list retrieval [SLV95]. For example, if the beginning or ending snapshot is required to satisfy certain constraints, only those sub-streams that begin or end with snapshot(s) satisfying the constraints are examined. Due to its high execution cost, examining all candidate substreams in the *AllSubStream(aStream)* set should always be the last resort for processing the stream predicate.

Possible indexing structures for retrieving similar streams (such as Fourier-transform [FRM94] and FastMap-based techniques [FL95, YJF98]) may be applied to search for substreams of similar trend. In the appendix A, a sub-stream indexing and retrieval technique based on suffix tree [Ste94] is presented as an initial effort on efficient retrieval of similar substream(s).

6.2 Knowledge-Based Query Processing Using Query Modification Techniques

Using the kernel class-based approach to implement the proposed constructs in KSTL has two problems: 1) **similar_to** operator is not implemented, 2) the **IS** and `approximate` methods need to be implemented inside every class which allows attributes to be queried using conceptual consturcts and approximate values. A knowledge-based approach is required to resolve the similar_to operator, conceptual and semantic terms since the interpretation of these conceptual constructs into value ranges of specific features are context- and user-dependent. Implementing this context- and user-dependent knowledge interpretation process into each individual subclass for the `IS` and `approximate` methods will make the knowledge maintenance difficult since specific value range for converting a conceptual term for a specific attribute should be built into methods of individual objects and the converted value ranges could be different depending on the query context and the types of users. The consistency across individual method is difficult to be verified. *Query modification* technique [CYC96], which modifies the extended query constructs into combinations of other query constructs directly supported by the query processing module, can be used to rewrite all the **IS** and **approximate** methods and *similar_to* operators based on a centralized knowledge repository to resolve this knowledge maintenance problem. Since all the proposed conceptual constructs are translated into value ranges of object features based on the same type of knowledge structures in a centralized knowledge repository [CYC96], the knowledge maintenance is simplified and the knowledge-based query modification process can be implemented in one module instead of being implemented in methods of every subclasses whose attributes can be queried using the proposed conceptual constructs. Inside the query modification module, relaxation control policy can also be easily implemented, maintained, and changed based on the query context and user types. This is not the case if the relaxation control policy need to be implemented inside the **IS** and **approximate** of every corresponding class.

The query translation/modification process is divided into *predicate analysis and feature selection*, and *knowledge-based content matching*. In the *predicate analysis and feature selection*, based on the target image, query context, and user type, the system analyzes and selects the relevant features and spatial relationships for processing these sub-parse-trees. In the *knowledge-based content matching*, the selected features, spatial relationships, and user types are used to match the TAH(s) that can resolve the conceptual terms, semantic spatial relationship operators, and/or similar-to predicates. The system uses the value ranges from the best matched TAH nodes to retrieve images from the image feature database.

To process the predicates with spatial relationship terms and/or conceptual terms, the spatial relationship terms and conceptual terms are used to select the TAH(s) and the TAH node(s) for processing these queries. The TAH nodes that yield best matching of the semantic terms and/or conceptual terms are used as the query conditions to retrieve the images. Their corresponding value ranges are used in the image retrieving.

For the *similar-to* predicate, the features and spatial relationships specified in the BASED_ON subclause are used to match with those of the target image. If no BASED_ON subclause is specified, then the query processor uses the default features and spatial relationships specified in the user model.

The selected features, spatial relationships, and user types are used to choose the TAH(s) for query processing. The selected TAHs are traversed to locate the node(s) whose value range is closest to that of the target image as the best matched TAH node. The set of images contained in that TAH node represents the set of images similar to the target image.

6.3 Flow of the KSTL Query Processing

The flow of processing the KSTL queries is presented in Figure 6.4 where both the kernel class-based technique and knowledge-based query modification technique are combined together to process KSTL queries. The process consists of 1) query parsing, 2) query execution, and 3) query relaxation. The KSTL query is parsed first and then the predicates are analyzed for execution. Predicates with built-in OQL constructs (e.g., $=$, $>$, etc.), are directly passed to the OQL query processor. For predicates with temporal, evolutionary, and stream constructs, the methods implemented in the corresponding kernel classes are invoked to resolve the predicates. For predicates with conceptual constructs, the knowledge-based query modification technique is applied to translate the conceptual predicates into specific value ranges of corresponding features based on the knowledge based (which is described in Chapter 4). The translated commands can be directly processed by the OQL query processors based on the existing methods, functions, and operators in OQL and the kernel classes. The initial query results can thus be retrieved from the database.

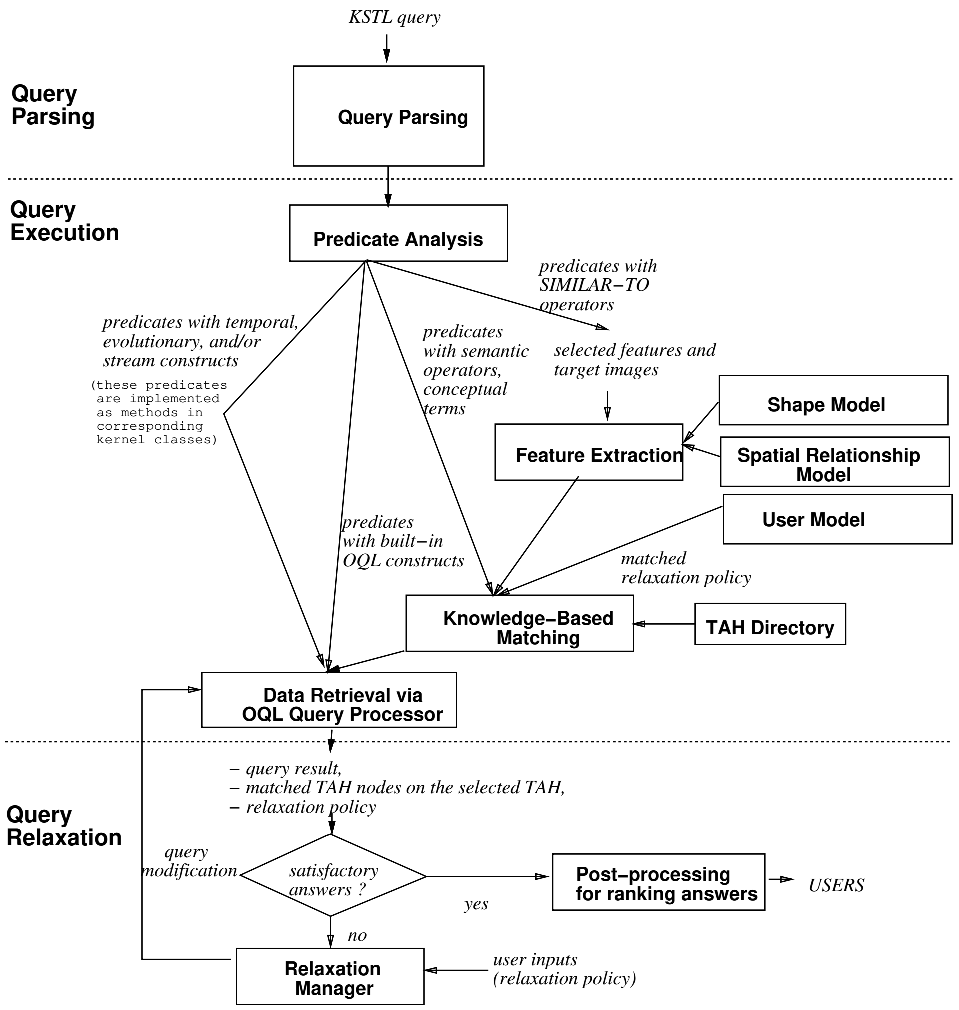


Figure 6.4: Knowledge-based query processing flow diagram for KSTL queries

During query relaxation, when no answer is returned or the returned answer is not satisfactory, the query conditions are relaxed in accordance with the relaxation specification. For the queries requiring relaxation, an appropriate TAH is selected to relax the query condition. The relaxation is performed by traversing up and down the selected TAH starting from the TAH nodes that best match the current query context. In every relaxation iteration, the query conditions are modified by the value from the TAH nodes. This relaxation process terminates when the answer set satisfies a given user requirement (e.g., number of similar images obtained, relaxation error, etc. [CC94, CYC96]). The returned images can be ranked based on a *relaxation error (RE)*, a similarity measure based on the weighted distance of the specified shape and spatial relationship features. Since the relaxation is conducted within a sub-parse-tree, it does not alter the query execution plan.

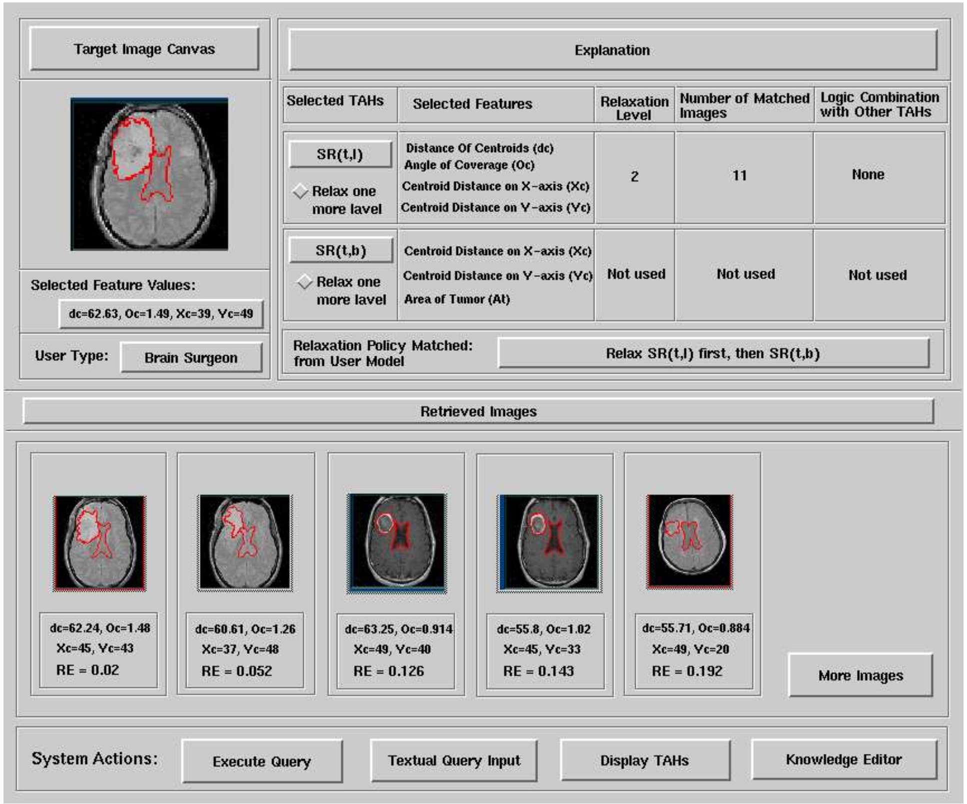


Figure 6.5: The graphical user interface (GUI) of the knowledge-based query answering showing the retrieved images that are most similar to the given target image

Figure 6.6 illustrates the query processing of a spatial predicate with *similar-to* operator for the target image as shown in Figure 6.5. The system allows user relaxation control input to overwrite the one provided by the user model. Since neither relaxation control policy nor `BASED_ON` subclause is provided by the user in this example, the relaxation control and matching policy specified in the user model are used (details of the matched user model are shown in Figure 4.2). Thus, SR(lesion,lateral_ventricle) is the first candidate TAH to be relaxed. Based on the TAH shown in Figure 3.5, the value ranges for retrieving similar images are:

$$(43.91 \leq SR(lesion, lateral\_ventricle).d_c \leq 71.31),$$
$$(0.85 \leq SR(lesion, lateral\_ventricle).\theta_c \leq 1.54),$$
$$(4.0 \leq SR(lesion, lateral\_ventricle).x_c \leq 49),$$
$$(-27 \leq SR(lesion, lateral\_ventricle).y_c \leq 57).$$

These conditions correspond to the value range of the TAH node two levels higher than the matched leaf node (see Figure 3.5). The retrieved images are shown and ranked with the *relaxation error* attached to each retrieved image (see Figure 6.5). There is an *explanation* window which displays the selected features and spatial relationships used for the matching, the relaxation level, and the number of matched images for this TAH node. During the relaxation process, if the relaxation of a TAH reaches a certain *relaxation error threshold*, then the system, according to the relaxation policy, selects another TAH for relaxation.

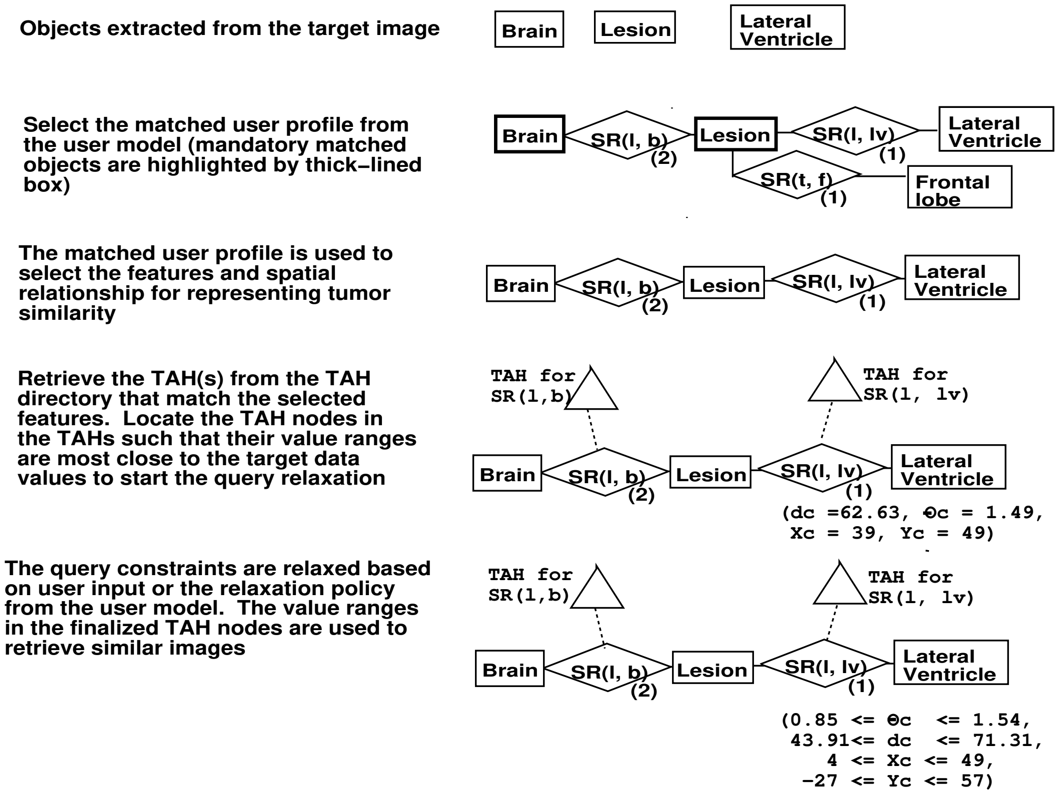


Figure 6.6: Processing steps for the query expressed in Figure 6.5 (the TAH for *lesion* `Nearby` *lateral ventricle* is shown in Figure 3.5)

CHAPTER 7

Conclusions

In this dissertation, techniques are proposed for image retrieval by feature and content with temporal, spatial, and conceptual constructs. Innovative techniques include: 1) a multi-layered integrated spatial and temporal data model that characterizes low-level image features (such as raw image data, and contours), abstract semantic image representations (including image objects and streams), and generic domain knowledge; 2) modeling of shape and spatial relationship content of image objects; 3) automatic analysis and classification of shape and spatial relationship features using MDISC clustering algorithm based on application-specific ontologies for knowledge-based query answering; 3) development of a knowledge-based spatial temporal query language, *KSTL*, which can experss and process queries with temporal, spatial, and conceptual image predicates based on temporal logic semantics; and 4) techniques for processing KSTL queries based on kernel class implementation and query modification technique using commercially availabel ODMG's OQL query processor.

APPENDIX A

Using Suffix Trees for Retrieving Sequences of Similar Patterns

[1]

A.1 Introduction

A sequence is an ordered collection of object snapshots (e.g., a sequence describing the temporal development of a group of tumors growing in the lung area of a patient). The trend of value change for objects' attribute(s) provides a *pattern* of the sequence. For a database system storing sequences, users should be able to retrieve sequences of specific patterns. Computing the similarity of patterns is important since it is often difficult to describe a target pattern precisely. In addition, patterns seldom exactly match one another as their minor difference often exist.

Moreover, in finding sequences with similar pattern to a given target sequence, all sub-sequences in the database should be searched since the satisfactory matches to the target sequence may reside anywhere in the data sequences. For

[1] A technique to index all sub-sequences in the databases for fast retrieval of similar sub-sequences is presented in this appendix. More verification and comparison are still needed. This work is part of the query processing and optimization to support the extended stream querying constructs in KSTL

a sequence S of m elements (e.g., $S = s_1, s_2, ..., s_m$), sequence S contains one subsequence of length m (i.e., S itself), two subsequences of length $m-1$ (i.e., $s_1, s_2, ..., s_{m-1}$ and $s_2, s_3, ..., s_m$), ..., and m subsequences of length 1 (i.e., s_1, s_2, ..., and s_m). The total number of subsequences of S is of $o(m^2)$ order. Given k data sequences in the database, candidate sequences to be compared with the target sequence is of the order $o(m^2k)$. Thus, these candidate sub-sequences should be indexed to improve the retrieval.

Fourier transformation [FRM94] or FastMap [FL95, YJF98] techniques convert the searched data sequences and their sub-sequences into a vector in a multi-dimensional feature space, and the converted vectors are indexed using spatial access methods such as R, R++, SS+ tree. However, these techniques do not preserve the snapshots' original ordering inside its sequences since a sequence is already converted into a vector in a multi-dimensional feature space. To remedy this shortcoming, suffix tree [Ste94], an indexing technique used in string matching, can be used to efficiently index and access all the sub-sequences of inserted data sequences. Suffix tree preserves the original ordering of indexed sequences in the indexing structure. Suffix tree is originally used to retrieve *exactly matched* sub-sequences to the target one. In this appendix, an algorithm is presented to retrieve similarly matched sub-sequences based on the suffix tree indexing.

A.2 Computing Pattern Similarity Using Dynamic Time Warping Technique

For two compared patterns S and T, let m and n be their length, and

$$S = s_1, s_2, ..., s_i, ..., s_m$$

$T = t_1, t_2, ..., t_j, ..., t_n$

The s_i and t_j $(1 \leq i \leq m, 1 \leq j \leq n)$ are individual snapshots of patterns S and T. A *pattern mapping* between patterns S and T is a possible mapping of all individual snapshots in S to those snapshots in T where a snapshot in S can be mapped to one or more snapshots in T. A pattern mapping between patterns S and T consists of a series of *snapshot mappings.* Let $\mathcal{N}$ be the set containing all the natural numbers (i.e., 1, 2, 3, ...). Let $I_S = \{i | 1 \leq i \leq m, i \in \mathcal{N}\}$. I_S is the set of integers used to index the snapshots in stream S. Similarly, $I_T = \{i | 1 \leq i \leq n\}$ is the set of integers used to index the snapshots in stream T. Let f be a function that maps a natural number to a snapshot in S, i.e., $f : \mathcal{N} \rightarrow I_S$. Similarly, let g be a function that maps a natural number to a snapshot in T, i.e., $g : \mathcal{N} \rightarrow I_T$. For example, the mapping of 1 to s_1, 2 to s_1, 3 to s_2, 4 to s_3, and 5 to s_5 means that $f(1) = 1$, $f(2) = 1$, $f(3) = 2$, $f(4) = 3$, and $f(5) = 5$. Let $\mathcal{F}$ be a set of functions defined as $\mathcal{F} = \{f | f : \mathcal{N} \rightarrow I_S\}$ and $\mathcal{G}$ be a set of functions defined as $\mathcal{G} = \{g | g : \mathcal{N} \rightarrow I_T\}$. Given a number $h \in \mathcal{N}$, snapshots $s_{f(h)}$ and $t_{g(h)}$ are mapped together as a snapshot mapping. Collecting all the snapshots mapping $(s_{f(h)}, t_{g(h)})$, a pattern mapping is defined which provides a specific way of mapping all the snapshots in S to all the snapshots in T based on functions f and h.

Constraints exist to restrict the functions f and g for the mapping among snapshots [BC96]. *Snapshot ordering* is the constraint that for any two neighboring snapshot mappings $(s_{f(h)}, t_{g(h)})$ and $(s_{f(h+1)}, t_{g(h+1)})$, the original ordering of the snapshots from a pattern should still be preserved (i.e. $f(h) \leq f(h+1)$ and $g(h) \leq g(h+1)$). *Continuity* is the constraint that no snapshot in either S or T can be skipped in the pattern mapping between S and T (i.e. $f(h+1) - f(h) \leq 1$

and $g(h+1) - g(h) \leq 1$ where $h \geq 1$). *Boundary constraint* is the constraint that the beginning (and ending) snapshots of S must be mapped to the beginning (and ending) snapshots of T.

Snapshot ordering, *continuity*, and *boundary constraints* are the three mandatory constraints for mapping patterns of time-based sequences since the embedded temporal ordering of snapshots in each pattern should be maintained (i.e., snapshot ordering) and no snapshot in the sequence could be skipped (i.e., continuity and boundary constraints).

Let a semantic function $d(s_{f(h)}, t_{g(h)})$ measure the *goodness of fit* of a snapshot mapping $(s_{f(h)}, t_{g(h)})$. For the pattern mapping between S and T based on functions f and g, the distance between S and T based on this particular pattern mapping can be defined as $\theta(S, T, f, g) = \Sigma_{h=1}^{||Dom(f)||}(d(s_{f(h)}, t_{g(h)})$ where $||Dom(f)||$ is the size of the domain for function f (i.e., the total number of integers where function f is defined). For all functions f and g in $\mathcal{F}$ and $\mathcal{G}$, the pattern mapping between S and T with the smallest value of $\theta(S, T, f, g)$ provides the measurement of how close the snapshots of S and T can be mapped to each other. $\mathcal{F}$ and $\mathcal{G}$ contain all the functions f and g that satisfy the snapshot ordering, continuity, and boundary constraints. This measurement is used as the *pattern similarity* between S and T. The smallest the value, the more similar the two patterns are. Thus, we define the similarity measure between S and T as

$$D(S, T) = min_{f \in \mathcal{F}, g \in \mathcal{G}}(\theta(S, T, f, g)).$$

Let $\gamma_{S,T}(i, j)$ be the pattern similarity between substrings $s_1 ... s_i$ and $t_1 ... t_j$ Thus, $\gamma_{S,T}(i, j) = D(s_1 ... s_i, t_1 ... t_j)$ and $\gamma_{S,T}(m, n) = D(S, T)$. Dynamic programming technique can be applied to optimize the computation of $D(S, T)$ based the following recurrence relation[BC96]:

$$\gamma_{S,T}(i,j) = d(s_i, t_j) + min(\gamma_{S,T}(i-1,j), \gamma_{S,T}(i,j-1), \gamma_{S,T}(i-1,j-1)).$$

This recurrence relation is based on the continuity and snapshot ordering constraints. Based on the boundary constraints, (s_1, t_1) is always the first snapshot mapping for any pattern mapping between S and T. Then for a *to-be-added* snapshot mapping $(s_{f(h+1)}, t_{g(h+1)})$, the candidate snapshots for $s_{f(h+1)}$ are restricted to $s_{f(h)}$ or $s_{(f(h)+1)}$ due to the continuity and snapshot ordering constraints. This is of the same case for selecting $t_{g(h+1)}$

For example, to derive the pattern similarity between S_1 in Figure A.2 and T_2 in Figure A.5, Table A.1 shows the values of $\gamma_{S_1,T_2}(i,j)$. Numerical difference of the mapped snapshot values is used as the semantic distance of a snapshot mapping in the computation, i.e. $d(v_1, v_2) = |v_1 - v_2|$. The highlighted cells in Table A.1 show the pattern mapping which defines the pattern similarity between S_1 and T_2. Figure A.1 shows how the snapshots are mapped in the resulted pattern mapping. Dynamic time warping technique requires $o(m^2)$ time complexity since there are $o(m^2)$ cells to be filled in the table for $\gamma_{S,T}(i,j)$. Here, m is the size of the target sequence.

Since the satisfactory matches to the target sequence may not always from the starting position of a data sequence, all sub-sequences of the data sequences should be searched to find sequences with similar pattern to the target pattern. Given a data sequence of length m, the total number of its sub-sequences is of $o(m^2)$ order. This search space need to be reduced to provide a feasible retrieving approach.

$s_8 = 40$	100	70	50	55	46	51	**60**
$s_7 = 30$	70	46	31	45	27	**30**	36
$s_6 = 20$	50	32	22	45	18	**16**	20
$s_5 = 10$	40	28	21	35	17	**12**	10
$s_4 = 20$	40	22	15	15	**6**	10	20
$s_3 = 30$	30	18	14	**5**	14	28	40
$s_2 = 20$	10	**4**	**5**	15	16	20	30
$s_1 = 10$	**0**	6	17	37	48	54	54
$S_1/\ T_2$ $\gamma_{S,T}(i,j)$	$t_1 = 10$	$t_2 = 16$	$t_3 = 21$	$t_4 = 30$	$t_5 = 21$	$t_6 = 16$	$t_7 = 10$

Table A.1: The cumulative distance table used in computing the semantic distance between T_2 in Figure 7.6 and S_1 in Figure 7.3. The highlighted cells are the snapshot mappings defining the pattern similarity for comparing S_1 with T_2

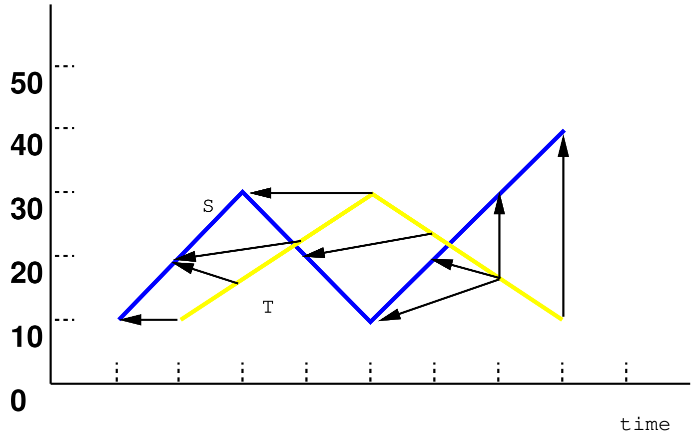


Figure A.1: The series of snapshot mappings which define the pattern similarity between S_1 with T_2

A.3 Using Suffix Trees for Retrieving Sequences of Similar Patterns

In string matching, a suffix tree [Ste94] provides the indexing structure to efficiently retrieve sub-strings which *exactly* match a given target string from data strings[2]. We propose to modify the suffix tree traversal algorithm for retrieving similar patterns.

A suffix of a pattern is a continuous sub-part of the pattern which ends with the last snapshot of the pattern (e.g., $s_3, s_4, ..., s_m$ and s_{m-2}, s_{m-1}, s_m are both suffixes of $s_1, s_2, s_3, s_4, ..., s_m$). Thus, a suffix is also a pattern. A prefix of a pattern is a continuous sub-part of the pattern which starts with the first snapshot of the pattern (e.g., $s_1, s_2, ..., s_{m-2}$ is a prefix of $s_1, s_2, ..., s_{m-2}, s_{m-1}, s_m$). Similarly, a prefix is also a pattern.

[2]In this appendix, strings and patterns are used interchangeably since they are all ordered collections of values and we are applying string matching technique to improve the pattern similarity retrieval. Similarly, *suffix* and *prefix* which is used to describe specific part of a string are also used to describe specific part of pattern.

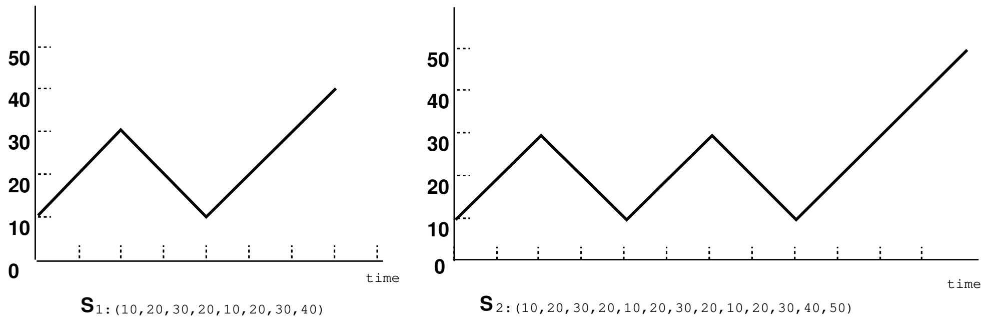
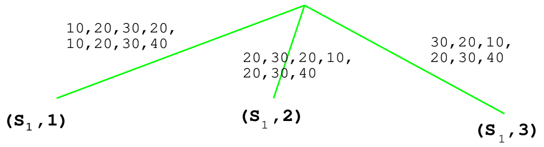


Figure A.2: Two sample sequences inserted into the suffix tree in Figure A.2

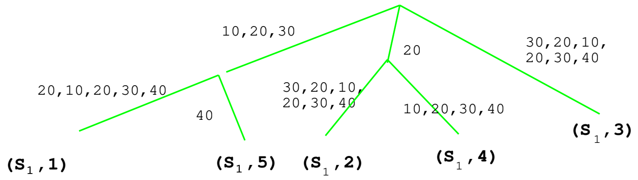
A suffix tree is constructed by inserting all the suffixes of data patterns into a tree structure. A leaf node represents an inserted suffix. Common prefixes of any two inserted suffixes share common edge(s). The label of an edge represents the shared sub-string. Common prefixes of any two suffixes can be found as the labels on their shared edge(s). For example, the process of constructing a suffix tree using S_1 and S_2 in Figure A.2 is illustrated in Figure A.3. In Figure A.3(a), the first three suffix of S_1 is inserted where no common prefix exist for these three suffixes and thus no common edge in the tree. In Figure A.3(b) when the next two suffixes of S_1 (i.e., 20,10,20,30,40 and 10,20,30,40) are inserted, the two common prefixes (i.e., 10,20,30 and 20) become the label of the shared branches. Figure A.3(c) illustrate the suffix tree after inserting all suffixes of S_1 and S_2. A leaf node in the suffix tree points to its represented suffix.

By traversing from the root, all the sub-sequences of the data sequences can be searched. Given a target sequence, the algorithm to locate an *exactly* matched sub-sequence is shown in Figure A.4 where the label for the root node is an empty sequence since it is the prefix of any sequence.

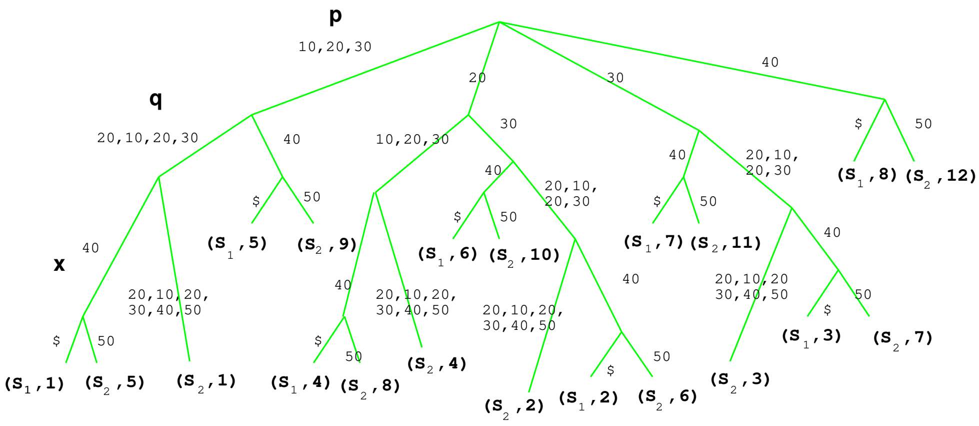
With k data sequences to be searched whose average length is l, there are $o(lk)$ suffixes; thus $o(lk)$ leaf nodes in the suffix tree. $o(log(lk))$ steps are required to



(a) the suffix tree after inserting (10,20,30,20,10,20,30,40), (20,30,20,10,20,30,40), (30,20,10,20,30,40)



(b) the suffix tree after inserting (20,10,20,30,40) and (10,20,30,40) into (a)



(c) the suffix tree after inserting all the suffixes for patterns in Figure 7.4

Figure A.3: The process of constructing a suffix tree as the two pattern S_1 and S_2 are inserted. Here \$ represents an empty string. In the leaf node, $(S_i, x), i = 1, 2$ means that the leaf node point to a suffix of S_i which starts at the x-th position of S_i.

Step 1: Starting from the root

Step 2: Select the child whose label matches a non-empty prefix of the target sequence. The matched prefix is removed from the target sequence

Step 3: Traverse to the selected child

Step 4: Goto Step 2 until a leaf node is reached the target sequence is exhausted

Step 5: Every leaf node under the node reached in Step 4 represents an occurrence of a matched subsequence

Figure A.4: The traversal algorithm for retrieving subsequences which exactly match the target sequence

locate the exactly matched sub-sequences from the $o(l^2k)$ candidate subsequences using suffix tree indexing since the complexity of searching the suffix tree is equal to the height of the suffix tree.

However, a fuzzy matching of the target pattern cannot be done by the algorithm in Figure A.4. For example, given T_2 in Figure A.5 as the target sequence, no matched sequences can be found from the suffix tree in Figure A.3 though similar matches do exist.

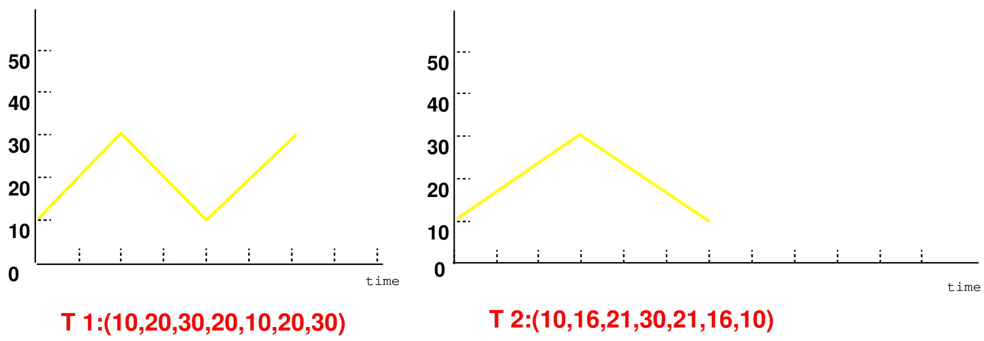


Figure A.5: Two target sequences.

A.3.1 Suffix Tree Traversal for Retrieving Similar Patterns

In this section, we propose a traversal algorithm which retrieve similarly matched patterns based on the suffix tree indexing structure. The dynamic time warping technique (detailed in Section A.2) is used to compute the pattern similarity between two sequences. Figure A.6 is an outline of our proposed algorithm for the similarity retrieval via suffix tree.

The proposed traversal is a pruning-based algorithm where a *cut-out threshold* is required to determine whether a branch should be visited or not. Usually users ask queries such as *"retrieve the best k answers similar to my target pattern"*. The value k can be used to derive the *cut-out threshold* for acquiring the best k answers. The steps **I.1** to **I.3** in Figure A.6 selects this initial cut-out threshold based on the value of k. This cut-out threshold is updated along the traversal process since the content in the buffer is continuously updated.

In Step **T.2** of the proposed algorithm, a function is needed to determine whether the child nodes of the currently investigated node e should be further explored. In the Step **T.4**, we need another function to determine whether any prefixes of the pattern inspected so far are patterns that are more similar to the target than those already selected in the buffer. These two functions should be computed based on the dynamic time warping technique.

A.3.1.1 Computing the Measurement for Pruning Branches

In step **T.2**, a function is needed to decide whether all candidate patterns under the currently investigated node e should be explored.

For a node in a suffix tree, its represented sequence is the sequence resulted from combining all the labels on the path from root to this node e. Let the sequence represented by a node e be S'. All the nodes under e have S' as their prefix. A function is defined to return the minimum similarity measurement between the target sequence T and all patterns with S' as their prefix. If even this minimum measurement exceeds the current cut-out threshold, then all the patterns under this investigated node e can not be the sequences that are more similar to the target sequence than those already selected in the buffer.

Let $S'(= s'_1, ..., s'_i, ..., s'_{m'})$ be a pattern, and $\mathcal{P}(S')$ be a set consists of all the patterns with S' as the prefix. Then the function to return the minimum similarity measurement between the target pattern T and those patterns starting with S' is defined as $D_{min}(S', T) = min\{D(y, T)|y \in \mathcal{P}(S')\}$.

Table of $\gamma_{S',T}(i, j)$ is used to compute $D_{min}(S', T)$. The smallest valued $\gamma(m', j)$ $(j \geq 1)$ is the value of $D_{min}(S', T)$ (where m' is the length of S'). The reason for the smallest valued $\gamma_{S',T}(m', j)$ to yield the minimum similarity measurement is based on the following fact. While appending $t_{j+1}...t_n$ (here n is the length of pattern T) to the end of S' as a new pattern to be compared with T, no additional semantic distance is introduced between T and this new constructed pattern since in this new pattern, all its snapshots after S' match the corresponding portion of T exactly. Therefore, as long as the distance among $s'_1...s'_{m'}$ and $t_1...t_j$ is minimized, we have the minimum semantic distance. Thus, during the suffix tree traversal, we need to compute the complete $\gamma_{S',T}$ values for S' and T in order to get the smallest valued $\gamma(m', j)$ $(j \geq 1)$ to compute $D_{min}(S', T)$. For example, when we are at the node **p** of Figure A.3 where $S' = 10, 20, 30$ and $T = 10, 16, 21, 30, 21, 16, 10$. Based on the values in Table A.1 the smallest valued $\gamma(3, j)$ is $\gamma(3, 4)$ which is 5. Thus, $D_{min}(S', T) = 5$.

The complexity of computing $D_{min}(S', T)$ is $o(max(m', n))^2)$ (where m' and n are the length of patterns S' and T) since in both cases, we only need to compute the $\gamma_{S',T}(i, j)$ where $1 \leq i, j, \leq max(m', n)$.

A.3.1.2 Acquiring More Similar Patterns During Suffix-Tree Traversal

In Step **T.4** of the proposed algorithm, we need to determine if the currently visited node has patterns more similar to the target pattern than those already selected in the buffer. Let $S'(= s'_1, ..., s'_i, ..., s'_{m'})$ be the pattern that the currently investigated node represents. Let the target pattern be T, and the cut-out threshold δ. We now define a function $\Omega(S', T, \delta)$ to return all the prefixes of S' whose distance to T is smaller than the current cut-out threshold δ [3].

Let $\mathcal{U}(S')$ be the set containing all the prefixes of S'. The function $\Omega(S', T, \delta)$ is defined as:

$\Omega(S', T, \delta) = \{y | D(y, T) < \delta, y \in \mathcal{U}(S')\}$.

The computation of the function $\Omega(S', T, \delta)$ is also based on the Table of $\gamma_{S',T}(i, j)$. Those $\gamma_{S',T}(i, n)(1 \leq i \leq m'$ where m' and n are the length of patterns S' and T) whose values are smaller than δ implied that the pattern $s'_1...s'_i$ is more similar to T than at least one pattern stored in the buffer since $D(s'_1...s'_i, T) = \gamma_{S',T}(i, n) < \delta$. For example, while traversing to the node **q** of the suffix tree in Figure A.3, $S' =$ 10, 20, 30, 20 10, 20, 30; $T = 10, 17, 21, 30, 21, 17, 10, 17, 21$. Let $\delta =$ 20, based on the values in Table A.1, only the last column of Table A.1 needs to be searched. The cells whose γ value in the last column is less than 20 are the

[3]Note that only the prefix (but not all the subsequences) of S' need to be compared with the target pattern since all suffixes are already inserted in the suffix tree.

selected pattern. Thus, in this example, we only have one answer.

A.3.2 Depth First Traversal for Optimizing the Distance Computation

The table computing $\gamma_{S',T}(i,j)$ is needed for computing $D_{min}(S',T)$ and $\Omega(S',T,\delta)$. Consider the situation that two data patterns S_1 and S_2 share a common prefix S', and both S_1 and S_2 are compared with the same target pattern T. The left lower portions of the tables $\gamma_{S_1,T}(i,j)$ and $\gamma_{S_2,T}(i,j)$, which both compute the pattern similarity between S' and the target pattern T are exactly the same. Depth first traversal strategy is thus used since all the patterns sharing common prefix are traversed one after the other. This makes those γ's of the shared prefix not to be repeatedly computed.

A.4 Future Research

Other techniques which can also retrieve similar sub-sequences from database sequences should be compared with the proposed searching techniques to verify the efficiency and effectiveness of the proposed algorithm. These competing techniques include:

- Fourier transformation-based approach [FRM94]
- FastMap-based approach [FL95, YJF98]
- Convolution: Taking the target sequence as a function and the compared data sequence as another function, convolution can be performed on these two functions. The location which produces maximum value for the con-

volution computation could represent a good matching between the target sequence and the data sequence.

Initial Steps

I.1 Randomly find k initial answers $(S_{b_1}, ..., S_{b_k})$,

I.2 Put these k answers in a buffer, rank these k answers based on $D(S_{b_i}, T), 1 \leq i \leq k$

I.3 Use the maximum value of $D(S_{b_i}, T), 1 \leq i \leq k$ as the initial cut-out threshold δ

$$\delta = max_{1 \leq i \leq k} D(S_{b_i}, T)$$

Traverse Down the Suffix Tree

T.1 Starting from root, investigate its children using depth first search traversal

T.2 Based on the cut-out threshold δ, check if the subtree under the currently investigated child node requires further visit

T.3 If the answer from **T.2** is yes, visit the node.

T.4 Among the sub-patterns of the currently investigated child node, check if there exist answers which are more similar to the target pattern than those always selected in the buffer

T.5 Update the buffer with any new answers acquired in **T.4**; update the cut-out threshold δ

T.6 Goto **T.1** until we traverse all satisfactory nodes

Figure A.6: Traversal algorithm for retrieving sequences similar to a target sequence T using suffix tree

APPENDIX B

The Conversion of a Temporal Logic Expression to a KSTL Query

A snapshot predicate is a *pure snapshot predicate* if it does not contain any modal operator. A snapshot predicate is a *simple temporal logic predicate* if it is a pure snapshot predicate or contains only one modal operator $\diamond$ or $\Box$. Let the temporal logic expression be $P = p_1 \ldots p_n$ where $p_i, 1 \leq i \leq n$ is a simple temporal logic expression. Let S be the set of all the streams for selection. During the conversion, the temporal logic expression P is translated into a KSTL statement with a FROM and WHERE clause. Since the temporal logic expression selects streams from S, the initial FROM clause is "FROM S as s".

All individual p_i, which expresses a temporal constraint on a snapshot or a substream, should be translated into corresponding KSTL predicates in the WHERE clause as follows:

case 1: If p_i is a pure snapshot predicate or a snapshot predicate with a modal operator $\diamond$, then the statement "x_i in s" is added to the FROM clause to bind a snapshot in s to x_i. The statement "$p_i(x)$" is added to the WHERE clause to express the constraint on the candidate snapshot. "$p_i(x)$" is the corresponding snapshot predicate of p_i expressed in KSTL.

case 2: If p_i is a snapshot predicate with a modal operator $\Box$, then the statement "$AllSubStream(s)$ as z_i" is added to the FROM clause to bind a candidate substream to z_i. The statement "forall y_i in z_i:$p_i(y)$" is added to the WHERE clause to express the constraint on the candidate substream. "$p_i(y)$" is the corresponding snapshot predicate of p_i expressed in KSTL.

The temporal relationship between the neighboring simple temporal logic expressions p_i and p_{i+1} $(1 \leq i \leq n-1)$ should now be added to the WHERE clause. For any two neighboring simple temporal logic expressions $p_i p_{i+1} (1 \leq i \leq n-1)$, constraints on their implied temporal relationship are added to the WHERE clause as follows:

case 1: When p_i is a pure snapshot predicate or a snapshot predicate with a $\diamond$ modal operator and p_{i+1} is a pure snapshot predicate, the snapshots satisfying p_i and p_{i+1} are immediate neighbors in the stream. Thus, the statement "$x_i.next == x_{i+1}$" is added to the WHERE clause to specify their temporal relationship.

case 2: When p_i is a pure snapshot predicate or a snapshot predicate with a $\diamond$ modal operator and p_{i+1} is snapshot predicate with $\diamond$ modal operator, the snapshot satisfying p_i appears before the snapshot satisfying p_{i+1} in the stream. The statement "$x_i.before(x_{i+1})$" is added to the WHERE clause to specify their temporal relationship.

case 3: When p_i is a pure snapshot predicate or a snapshot predicate with a $\diamond$ modal operator and p_{i+1} is snapshot predicate with a $\Box$ modal operator, the snapshot satisfying p_i and the first snapshot of the substream satisfying p_{i+1} are immediate neighbors in the stream. The statement

"$x_i.next == first(z_{i+1})$" is added to the WHERE clause to specify their temporal relationship.

case 4: When p_i is a snapshot predicate with a $\Box$ modal operator and p_{i+1} is a pure snapshot predicate, the last snapshot of the substream satisfying p_i and the snapshot satisfying p_{i+1} are immediate neighbors in the stream. The statement "$last(z_i).next == x_{i+1}$" is added to the WHERE clause to specify their temporal relationship.

case 5: When p_i is a snapshot predicate with a $\Box$ modal operator and p_{i+1} is a snapshot predicate with a $\Diamond$ modal operator, the last snapshot of the substream satisfying p_i is occurs before the snapshot satisfying p_{i+1} in the stream. The statement "$last(z_i).before(x_{i+1})$" is added to the WHERE clause to specify their temporal relationship.

case 6: When p_i is a snapshot predicate with a $\Box$ modal operator and p_{i+1} is a snapshot predicate with a $\Box$ modal operator, the last snapshot of the substream satisfying p_i and the first snapshot of the substream satisfying p_{i+1} are immediate neighbors in the stream. The statement "$last(z_i).next == first((z_{i+1})$" is added to the WHERE clause to specify their temporal relationship.

Note that p_1 and p_n express the boundary conditions that a selected stream should satisfy. These boundary conditions should also be translated into KSTL as follows:

case 1: When p_1 is a snapshot predicate with a $\Box$ modal operator, the statement "$first(z_1) == first(s)$" is added to the WHERE clause to specify the beginning condition of the stream s.

case 2: When p_1 is a pure snapshot predicate, the statement "x_1) $== first(s)$" is added to the WHERE clause to specify the beginning condition of the stream s.

case 3: When p_n is a pure snapshot predicate or snapshot predicate with a $\diamond$ modal operator, the statement "$x_n == last(s)$" is added to the WHERE clause to specify the ending condition of the stream s.

case 4: When p_n is a snapshot predicate with a $\diamond$ modal operator, the statement "$last(y_n) == last(s)$" is added to the WHERE clause to specify the ending condition of the stream s.

Finally, relationships between x_i's (e.g., INTERVAL[x_2,x_1].approximate(T) where T is an interval length specified by users) are added into the WHERE clause. Note that these relationships cannot be expressed inside a temporal logic expression due to unclear variable binding in a temporal logic expression.

References

[BC96] D. J. Berndt and J. Clifford. "Finding Patterns in Time Series: A Dynamic Programming Approach." In U. M. Fayyad, G. Piatetsky-Shapiro, et al., editors, *Advances in Knowledge Discovery and Data Mining*, 1996.

[BPJ93] J. R. Bach, S. Paul, and R. Jain. "A visual information management system for the interactive retrieval of faces." *IEEE Transaction on Knowledge and Data Engineering*, October 1993.

[Car94] R. Cartell, editor. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, 1994.

[CC94] W. W. Chu and Q. Chen. "A Structured Approach for Cooperative Query Answering." *IEEE Transactions on Knowledge and Data Engineering*, **6**(5), October 1994.

[CCH96] W. W. Chu, K. Chiang, C. C. Hsu, and H. Yau. "An error-based conceptual clustering method for providing approximate query answers." *Communications of ACM, Virtual Extension Edition (http://www.acm.org/cacm/extension)*, December 1996.

[CCK84] M. Chock, A. F. Cardenas, and A. Klinger. "Database Structure and Manipulation Capabilities of a Picture Database Management System (PICDMS)." *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **Vol. PAMI-6**(No. 4), July 1984.

[CHI98] W. W. Chu, C. Hsu, I. T. Ieong, and R. Taira. "Content-Based Image Retrieval using Metadata and Relaxation Techniques." to appear as a book chapter in "Managing Multimedia Data: Using Metadata to Integrate and Apply Digital Data" edited by W. Klas and A. Sheth, McGraw Hill, 1997, 1998.

[CIT94] W. W. Chu, I. T. Ieong, and R. K. Taira. "A Semantic Modeling Approach for Image Retrieval by Content." *VLDB journal*, 1994.

[CYC96] W. W. Chu, H. Yang, K. Chiang, et al. "A Scalable and Extensible Cooperative Information System." *Journal of Intelligent Information Systems*, **6(6)**, 1996.

[CYD88] S. K. Chang, C. W. Yan, and D. C. Dimitroff. "An Intelligent Image Database System." *IEEE Transaction on Software Engineering*, May 1988.

[DC96] J. D. N. Dionisio and A. F. Cardenas. "MQuery: A Visual Query Language for Multimedia, Timeline, and Si mulation Data." *Journal of Visual Languages and Computing*, 1996. Special issue on *Image and Video Databases: Visual Browsing, Querying, and Retrieval*.

[DSB95] H. Duong, L. Sarazin, P. Bourgouin, and J. L. Vezina. "Magnetic resonance imaging of lateral ventricular tumours." *Canadian Association of Radiologists Journal*, **46(6)**:434–442, Dec 1995.

[Dv93] D. Daneels, D. van Campenhout, et al. "Interactive outlineing: An improved approach using active contours." In *Image and Video Storage and Retrieval, SPIE*, 1993.

[FL95] C. Faloutsos and K.-I. Lin. "FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets." In *ACM SIGMOD*, 1995.

[FRM94] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. "Fast Subsequence Matching in Time-Series Databases." In *ACM SIGMOD*, 1994.

[Gab89] D. Gabbay. "The Declarative Past and Imperative Future Temporal Logic in Specification: Altrincham Workshop." In *LNCS 398*. Springer-Verlag, 1989.

[GJ97] A. Gupta and R. Jain. "Visual Information Retrieval." *CACM May*, 1997.

[GM90] W. I. Grosky and R. Mehrotra. "Index-Based Object Recognition in Pictorial Data Management." *Comput. Vis. Graph. and Image Proc.*, **Vol. 52**(No. 3):416–436, 1990.

[GM91] D. Gabbay and P. McBrien. "Temporal Logic and Historical Databases." In *Conference on Very Large Data Bases*, 1991.

[Gri90] W. E. L. Grimson. *Object Recognition By Computer: the Role of Geometric Constraints*. MIT Press, 1990.

[GWJ91] A. Gupta, T. Weymouth, and R. Jain. "Semantic Queries with Pictures: the VIMSYS Model." In Guy M. Lohman, Amilcar Sernadas, and Rafael Camps, editors, *Proceedings of the 17th International Conference on Very Large Databases*, pp. 69–79, Barcelona, Spain, September 1991. Morgan Kaufman.

[HC97] C. Hsu and W. W. Chu. "A Knowledge-Based Object-Oriented Language for Temporal Image Databases." Submitted to ACM SIGMOD 1998, 1997.

[HCT96] C. Hsu, W. W. Chu, and R. K. Taira. "A Knowledge-Based Approach for Retrieving Images By Content." *IEEE Transactions on Knowledge and Data Engineering*, Aug. 1996.

[KC88] W. Kim and H.T. Chou. "Versions of Schema for Object-Oriented Databases." In *Proc. of the 14th VLDB*, 1988.

[Lev85] M. D. Levine. *Vision in Man and Machine*. McGraw Hill Publishing Co., Inc., 1985.

[LG93] T. Little and A. Ghafoor. "Interval-Based Conceptual Models for Time-Dependent Multimedia Data." *IEEE Transactions on Knowledge and Data Engineering*, Aug. 1993.

[LTS95] B.J. Liu, R.K. Taira, J. Shim, and P. Keaton. "Automatic Segmentation of Bones from Digital Hand Radiographs." In *Proceedings of the SPIE: Medical Imaging, Image Processing*, volume 2434, pp. 659–669, February 1995.

[MG95] R. Mehrotra and J. Gary. "Similar-SHape Retrieval In Shape Data Management." *IEEE Computer*, September 1995.

[Mil95] G. Miller. "WordNet: A Lexical Database for English." *Communications of CACM*, November 1995.

[NBE93] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, P. Yanker, C. Faloutsos, and G. Taubin. "The QBIC project: Querying images by content using color, texture, and shape." In *Storage and Retrieval for Images and Video Databases, SPIE*, 1993.

[OM88] J. Orenstein and F. A. Manola. "PROBE Spatial Data Modeling and Query Processing in an Image Database Application." *IEEE Transactions on Software Engineering*, May 1988.

[RFS88] N. Roussopoulos, C. Faloutsos, and T. Sellis. "An Efficient Pictorial Database System for PSQL." *IEEE Transaction on Software Engineering*, May 1988.

[Ric92] J. Richardson. "Supporting Lists in a Data Model (A Timely Approach)." In *ACM SIGMOD 1992*, 1992.

[RS91] F. Rabitti and P. Savino. "Image Query Processing based on Multi-level Signatures." In *Proceedings of ACM Information Retrieval*, 1991.

[RS92] F. Rabitti and R. Savino. "An Information Retrieval Approach for Image Databases." In *Proceedings of 18th VLDB Conference*, 1992.

[SHB93] M. Sonka, V. Hlavac, and R. Boyle. *Image, Processing, Analysis and Machine Vision*, volume 1. Chapman & Hall Computing, 1993.

[SLV95] B. Subramanian, T. W. Leung, S. L. Vandenberg, and S. B. Zdonik. "The AQUA Approach to Querying Lists and Trees in Object-Oriented Databases." In *IEEE International Conference on Data Engineering*, 1995.

[Sno87] R. Snodgrass. "The Temporal Query Language TQUEL." *ACM TODS*, 1987.

[Ste94] G. A. Stephen. *String Searching Algorithms*. World Scientific, 1994.

[US92] U.S. Department of Health and Human Services, National Institutes of Health, National Library of Medicine. *UMLS Knowledge Sources*, August 1992.

[WA95] R. Wasserman, R. Acharya, et al. "Multimodality Tumor Delineation via Fuzzy Fusion and Deformable Modelling." In *Proceedings of the SPIE, Medical Image: Image Processing*, 1995.

[WLK92] A. J. Worth, S. Lehar, and D. N. Kennedy. "A Recurrent Cooperative/Competitive Field for Segmentation of Magnetic Resonance Brain Images." *IEEE Transactions on Knowledge and Data Engineering*, **4**(2):156–161, April 1992.

[YJF98] B.-K. Yi, H. V. Jagadish, and C. Faloutsos. "Efficient Retrieval of Similar Time Sequences Under Time Warping." In *IEEE International Conference of Data Engineering*, 1998.