

Graph Database Systems for Genomics

Mark Graves, Ellen R. Bergeman, Charles B. Lawrence

Department of Cell Biology, Baylor College of Medicine

Correspondence:

Mark Graves
Department of Cell Biology
Baylor College of Medicine
One Baylor Plaza
Houston, TX 77030
mgraves@bcm.tmc.edu (e-mail)
713-798-8271 (voice)
713-798-3759 (fax)

Ellen R. Bergeman
Department of Cell Biology
Baylor College of Medicine
erb@bcm.tmc.edu (e-mail)
713-798-8105 (voice)

Charles B. Lawrence
Department of Cell Biology
Baylor College of Medicine
chas@bcm.tmc.edu (e-mail)
713-798-6226 (voice)

Abstract

Genome databases have specific requirements which limit the usefulness of some database management systems. By using more appropriate database technology, a database system can be developed for genome data. We have developed a data representation based on graph theory which captures the highly interconnected structure of genome data. Graphs are a language which can be tailored for describing genomic information, and we develop a data model based on graphs which serves as the foundation of a graph database management system.

1. Introduction

Genome project data is being generated at a rapidly increasing rate as the project transitions from technology development to data production. Techniques such as polymerase chain reaction (PCR) have simplified the creation of genome maps from a nearly impossible task to one that can be made semi-automatic. Unfortunately, as biologists have discovered new ways to explore the genes which make us who we are, genome informatics is struggling to keep up with biology and information management technology.

Laboratory researchers are producing data which must be accessible to them. In the past, laboratory notebooks worked well for private data, and simple databases worked to store public data in a standard format which was reasonably useful. But now laboratory notebooks are too awkward to store the quantities of data being generated and simple databases do not allow the flexibility needed for the rapidly progressing laboratory processes.

A database management system must be developed which can propel genome research forward, rather than holding it back. As more genomic data is generated, data analysis will become more essential, and we must be ready to support the analysis with appropriate database management tools.

Our approach to providing database support is to examine the requirements of genome databases, tailor a data representation language to those requirements, develop a data model to organize that data within a database, and build a database management system to support the data model.

2. Genome Databases

Genome databases are difficult to build using current technology. Although database systems are sufficient to capture the quantity of data being generated, they are inadequate for the rapidly changing types of data, its complexity, and imprecise definition. Other areas of computer science have developed means of dealing with these inadequacies, but no single previously developed system incorporates the needed functionality. We have developed a database management system which includes this functionality.

2.1 Requirements

Genomics is the discipline of constructing and interpreting genomic maps -- maps which contain all the genetic material of an organism. Genomic data is typified by:

1. large, rapidly growing quantities,
2. rapidly changing types,
3. complex, interconnected structures, and
4. imprecise definitions.

Genomics requires database systems to capture genome data. The quantity of data has overwhelmed what can be stored in traditional laboratory notebooks, and the amount of data being generated is growing at an exponential rate. New, automated laboratory techniques are likely to continue increasing the rate of data generation. Database systems are currently capable of storing the data being generated by genome projects and will continue to be pushed by other technologies, such as multi-media systems and astronomy satellites, which generate much larger quantities of data than biology. However, current database systems are not completely adequate.

Genomics is changing rapidly. Because genetics research is advancing at a rapid rate, it is important that genomic data be represented in a flexible framework which can be easily extended. The type of data being generated in laboratories changes frequently. Many labs use techniques, such as polymerase chain reaction, which were discovered only a few years ago, and techniques which were a great advance in science a decade ago, such as using restriction fragment length polymorphisms to generate maps, are now obsolete. Databases have never been developed to capture data from such a rapidly changing science, and techniques to speed up the development of databases are still immature. An additional problem is that the changing concepts are not simple.

Genome data is highly-interconnected and has a complex structure which is difficult to capture in any current database system. Genetic information consists of large numbers of highly interconnected constructs: genes, regulatory mechanisms, introns, exons, alternative splicings, binding sites, tissue-specific expression, polymorphisms, homologous regions, etc. Each construct is not primitive, but is defined by its relationships with other constructs, for example a gene is not *a priori* a gene, but is a region of DNA with some sequence which is translated into some protein and expressed in some cell. The intrinsic connectivity of genome data must be captured in a database.

Genomic concepts are not logically and precisely defined. Unfortunately, most data representations assume the existence of precise, logical definitions which causes difficulties in using them for biological databases. In some domains, a set of undefined primitives may be defined upon which all other definitions are built. In an experimental science, such as biology, the concepts are defined over time and the process of describing the concepts, such as genes, promoters, and regulatory mechanisms, embody much of the science. Biologists are comfortable with this uncertainty, but computers (and computer scientists) are not.

2.2 Available Technologies

Other areas of computer science have addressed some of the requirements for genome databases. Database systems provide support for large amounts of data; software engineering addresses the need to build systems which frequently change; artificial intelligence has contributed languages to represent complex, highly-connected data; and computational linguistics has developed ways of capturing the imprecise definitions of nature in a computational language.

The rapid development of extensible software is one approach to developing systems which frequently change, and software engineering techniques have been developed to support it. Three relevant software engineering techniques are rapid prototyping, domain analysis, and object-oriented design. Rapid prototyping is a mechanism for delivering functional, but incomplete, systems to users to obtain feedback before completing development [1]. In some domains, rapid prototyping may not speed up total development time, but genetics is changing so rapidly that prototypes must be delivered because traditional programming techniques may produce systems which are obsolete before they are completed. Domain analysis documents concepts in the domain in a way they can be used to develop multiple software systems [2]. Originally, domain analysis was developed to reduce duplicated effort in large software projects, but it is useful in genetics because the rapid turnover of systems require that as much as possible of previous system design be reused. Object-oriented design complements domain analysis by striving for modularity and isolation of the actual implementation [3]. Objects can be reused directly in the next implemented system.

Knowledge representation formalisms from artificial intelligence can describe complex, highly-interconnected concepts. One useful representation for genome data is semantic networks. Se-

semantic networks capture the meaning of concepts by describing how they are related to other concepts. Semantic networks represent concepts and relationships as a graph with labeled nodes and arcs. Each concept is defined by its relationship to other concepts. Semantic network formalisms are used in two genome database projects, LabBase [4] and ACeDB [5].

Computational linguistics provides formal languages to describe vague, ambiguous, and conflicting concepts. Semantic networks are not sufficient to represent concepts which are not precisely and logically defined. One of the most difficult representation problem which has been tackled by computer science is to capture the semantics of natural language. Computational linguists have extended formalisms such as semantic networks to capture the impreciseness of natural language. Luckily, some of these formalisms are also useful for capturing genome data.

2.3 Our Solution

No previous system incorporates the functionality specified in the four requirements. Some systems do incorporate two of the four, such as graph databases, programming languages developed for computational linguists, neural networks, and extensible databases. If an existing system had incorporated three of the four, a possible alternative would be to extend that system. However, no existing system provided enough of a foundation upon which to build, so we developed from scratch an extensible, graph database management system based on computational linguistic formalisms.

Incorporating the available technologies into one database system is simplified by treating graphs as the basic representation. Graphs capture the intrinsic connectivity of genome relationships, and they are a comfortable communication medium for biologists and computer scientists. We have developed a genome graph language, called GGL, which represents genome data as a graph. A graph in GGL is a collection of vertices and edges where vertices represent the concepts and relationships of genetics, and edges describe the connections between them. A graph which shows many of the concepts and relationships of genetics discussed in the issue is given in Fig. 1. The graph shows general concepts, such as genome and species, as well as more specific ones, such as protein product and mRNA. Different databases represent parts of the graph at different levels of detail. For example, some databases represent sequence as a string of the four characters A, G, C, and T, while GSDB [6] contains much more information.

Graph database systems have been developed (as discussed in section 4.2), and they can be made extensible by adding mechanisms to change the graphs that store data. We have developed a graph database management system based on our graph language GGL which facilitates the storage of genome data in a database. Graphs are a foundation of semantic networks which have been useful in previous genome database projects. More recent work in computational linguistics has updated semantic network formalisms [7,8], and these can be used to capture the impreciseness of genomic definitions.

3. Graph Representation

In developing any complex, data-intensive system, the most crucial technical decision to make is the choice of data representation. In developing a graph database management system to support genome databases, the most appropriate data representation is a graph. However, there are many kinds of graphs and many graph languages available from computer science. Genomic data must be examined more closely to distinguish the kinds of graphs required to capture the structure

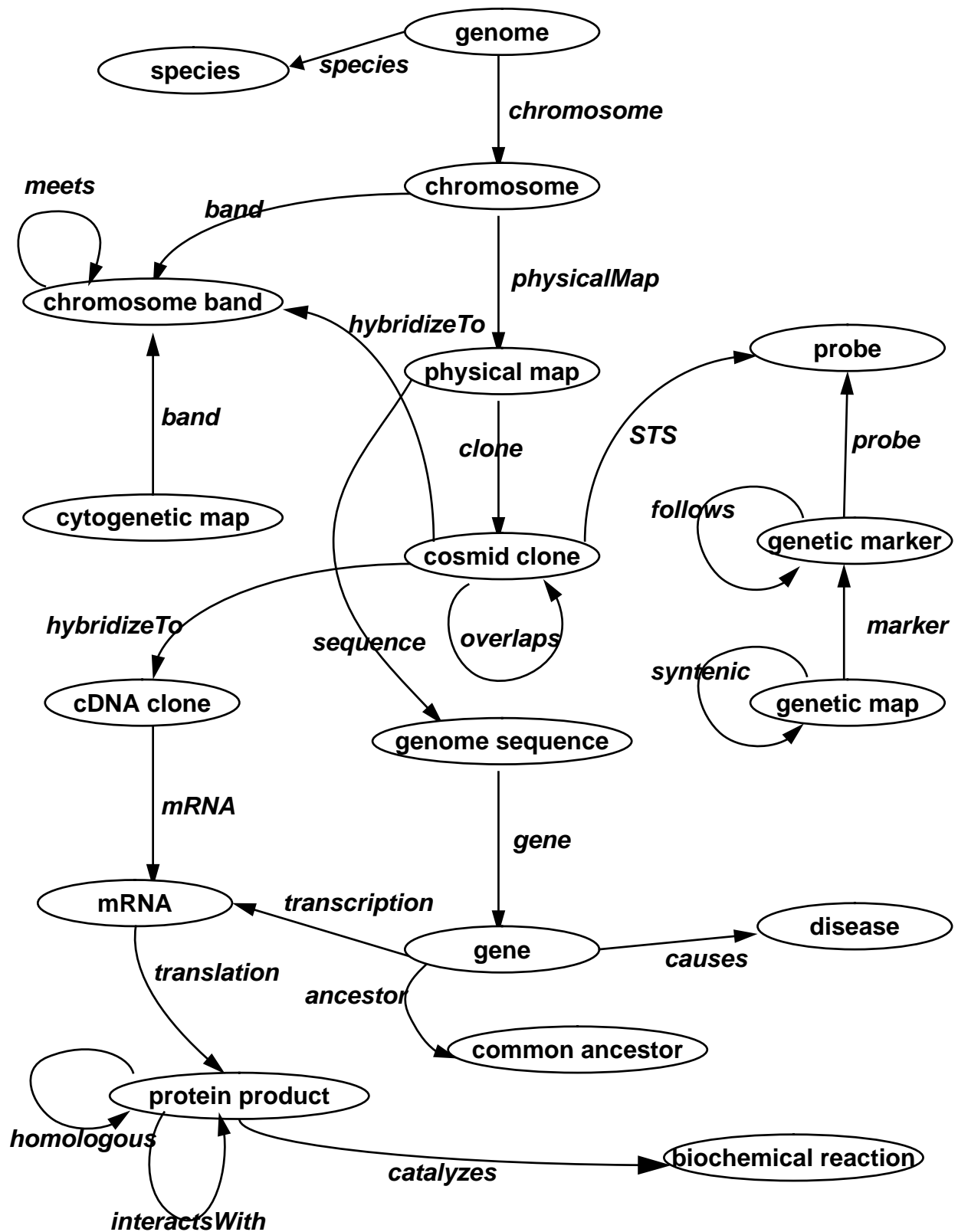


Figure 1: A graph showing many concepts and relationships in genetics.

of the data, and then a graph language should be chosen which can represent the data without unnecessary complexity.

The four characteristics of genome data listed in Section 2.1 are addressed by describing the representation of the data as a graph language (this section) and then developing a data model based on that language (Section 4). Our graph representation language addresses the last three characteristics -- rapidly changing types, interconnected structure and imprecise definition of genome data -- and our data model addresses the first characteristic by describing how large, rapidly growing quantities of data may be stored in a database.

3.1 Requirements

A representation language designed to capture genome data should:

1. emphasize concepts and relationships equally,
2. describe complex, regular structures,
3. support cyclic graphs, and
4. be based on binary relationships.

A representation language which emphasizes concepts and relationships equally simplifies changing the type of data in the database. Most concepts in genetics are defined in terms of their relationships to other concepts. A representation language which does not create an arbitrary distinction between concepts and relationships allows a concept to grow from a simple, undefined concept to one defined by multiple complex relationships.

Genetic relationships are determined by experimental evidence. In molecular biology, most of the science is based on indirect observations, and the results of these experiments are relationships between the reagents used in the experiment and the result found, i.e., hybridization results, order information or function. There are few primitive concepts in genetics, other than the biochemistry, so most of the results are relationships between relationships between relationships, etc. The relationships form a graph-like structure.

A representation language which describes complex, regular structures and supports cyclic graphs captures the complex, interconnected structure of genome data. Genomic data has a complex structure which can often be decomposed into regular substructures. In computer science, data structures such as trees are built from simple node and leaves. In molecular biology, the three-dimensional structure of a chromosome contains coils of coils. Although these substructures may be described as graphs, more efficient mechanisms may take advantage of the regularities. Genomics contains several different kinds of graphs which occur frequently, such as trees and cyclic graphs. These graphs may be classified by their structure.

Two useful ways to subdivide graphs are whether they contain cycles, and whether they are rooted. This classification generates four types of graphs: cyclic graphs, directed acyclic graphs, rational trees, and trees. The graph-theoretic definition of a graph is a collection of vertices and edges, where each edge connects two vertices in the graph.

cyclic graph is a graph where some vertex would be repeated if edges were followed in a path from first node to second node.

directed acyclic graph is a graph which is not cyclic.

rational tree is a graph with one vertex from which all other vertices may be reached.

tree is a rational tree which is not cyclic.

When data with a cyclic graph structure is coerced into an inappropriate structure, much information is lost. One of the limitations of current database technology in developing genome databases is the inability to model the cyclic graph structure of genome data. Coercing cyclic graphs into a linear or tree-like structure will cause some information to be lost and make development of systems on top of that representation more difficult. To store data in existing databases, the graph structure of data must be removed. The removal of structure biases the data and the incompleteness of information can make some tasks difficult to perform. It also becomes difficult to later isolate the structure from the behavior of the application when the changing science necessitates a change in the structure. A useful genome database must be able to represent the complex structure. Representing genome data as a graph helps alleviate the loss of structural information in genome databases.

Graphs are pervasive within genomics. Cyclic graphs occur in gene regulation [9], metabolic pathways [10,11,12], chemical structure, map order with uncertainty [13], and homology relationships between species. Representing genomics requires more cyclic graphs than most other areas in which databases are used, and this limits the usefulness of previous systems. Most computational systems are not designed to handle cyclic data structures (recursive definitions) well. Other types of graphs are also common within genomics and can be incorporated within graph-based systems. Trees are useful in representing phylogenetic relations [14] and the organization of DNA coils. Robbins [15] proposes a database be described as a collection of truncated rational trees, and ACeDB schemas are represented using linked trees which form directed acyclic graphs.

A representation language based on binary relationships supports the imprecise definition of genomic data. Because most of the aspects of each genome relationship are independent of each other, it is useful to decompose (fully normalize) the relationships into binary relations, i.e., those with an arity of two, which describe one aspect of a concept. Binary relations can be combined with other binary relations to describe concepts which may not have been considered when the database was originally developed. For example, an oligonucleotide may have originally been intended to be a PCR primer, but now may be considered to be a STS or non-polymorphic marker within the database.

3.2 Available Technologies

After recognizing the graph structure within genome data, there are many options from which to choose a graph representation. Graphs are pervasive in computer science and are the basis of several representation languages, including: semantic networks, ER diagrams, functional data models, semantic data models, terminological description languages, attribute value formalisms, finite-state automata, Petri nets, function nets, data flow diagrams, conceptual graphs, ψ -types, feature structures, and feature logics. An appropriate graph language must be found to capture genome data. We incorporate several features of existing graph languages in our genome graph language.

3.3 Our Solution

When examining the characteristics of each language, we discovered a small, concise set of features which would meet the requirements. None of the existing graph languages could treat concepts and relationships similarly, capture complex, regular graphs and cyclic graphs, and build on binary relationships. Those languages which were close often added functionality which was

not needed. Many existing graph languages seem almost identical until theoretical investigation points out the differences, but it seemed important that the language used to describe genome data fit the domain well. GGL is a graph representation language which is simpler than most existing graph languages and is better for describing genomic data.

GGL has graph theory as its foundation and extends the mathematical definition of a graph to make them more useful for representing genome data. Graphs are defined as in graph theory as a collection of vertices and edges where:

- Vertices are the nodes of the graph. In GGL, vertices model the simple concepts and n-ary relations of the genetics.
- Edges connect two vertices. There cannot be multiple edges between two vertices.

The basic definition of a graph must be extended by adding identifiers to make it more useful for representing genome data.

- Labels on edges define the type of relationship between two vertices.
- Symbols are nodes in the graph which identify external objects.

The basic graph language describes a graph such as the one shown in Figure 2. Because sche-

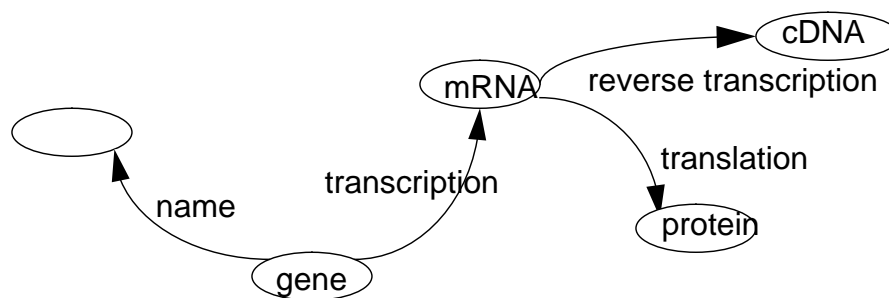


Figure 2: A Schema in the Genome Graph Language

mas describe graphs in the database, they label the nodes which are not symbols.

Edges are identified by adding labels to them, and symbols are a subclass of nodes which are used to identify some of the nodes. However, GGL does not include labels on all the nodes. Some of the nodes must be labeled, but not all of them need to be. Labels are needed for the name of a person, location of a filter or test tube, and other external identifiers. Many nodes do not need external identifiers and requiring them makes development more difficult. Many early graph languages added labels to the nodes, but some later graph languages distinguish between nodes that refer to external objects and those that are internal to the database. Symbols are defined as nodes in the graph that have no outgoing arcs and which have an unique identifier. The identifier is a string and is used to relate the data in the database to external objects.

Although GGL is a simple language, we have found it useful for representing a variety of genome objects. The next step is to modify GGL to store large amounts of data. This is accomplished by describing it as a data model.

4. Graph Data Model

A representation language, such as GGL, is useful for describing the concepts of genetics, but is not sufficient to describe how it would be used in a database. For example, we need to also describe how graphs are built, what operations can be performed on graphs, and any other constraints on what a graph in the database can look like. There are other ways to describe a

representation language, but describing it as a data model is the most common and useful mechanism within the database area.

4.1 Requirements

A data model may be defined by specifying its data types, operators, and constraints. Data types describe the basic building blocks of data. For example, that the data consists of numbers, strings, and sets. The operators specify how data can be manipulated. For example, it is valid to add an element to a set or add two numbers together to get a third, but there is not usually an operator to delete a digit from the middle of a number. Constraints define the restrictions on which instances of the data types can legally occur in the database. For example, a possible constraint is that a “set” can consist of either numbers or strings but not both.

Relational databases were the first to be specified in terms of a data model [16,17]. The data type for the relational model is the relation. The eight operators are select, project, join, product, union, difference, intersection and division. There are two database-independent integrity constraints: (1) No component of the primary key of a base relation is allowed to accept nulls; (2) The database must not contain any unmatched foreign key values.

GGL already specifies part of the data model. The data types are vertices, edges, and symbols. There is a constraint that symbols are not the source node of an edge.

4.2 Available Technologies

During the past few years, data models have been developed which take advantage of graph-based representations to model complex structure. Other systems have been developed to better isolate the user from implementation details, such as logic data models, which are defined using mathematical logic and often use a subset of first-order predicate logic. Although graph-theoretic data models may appear similar to network or semantic data models, they are more closely related to logic data models [18,19].

Previous data models have incorporated many ideas which originated in artificial intelligence, such as semantic networks, inheritance, and active databases. Other ideas have originated in artificial intelligence and been refined within programming languages before being incorporated in databases such as objects and logic programming. Graph data models are no exception and are based on much of the same foundation within artificial intelligence. Graph data models also draw upon the tremendous body of research in graph theory, which is one of the larger areas of research in mathematics and computer science.

Graph data models define the graph representation for database systems. All graph data models have as their foundation the mathematical definition of a graph as a collection of vertices and edges. The data models are usually used to support an application, such as visual querying [20], end-user interfaces [21], hypertext systems [22], or natural language processing [18]. Unlike the relational data model which has one incarnation, there are several extensions to the definition of graphs which form foundations for different graph data models. For example, most graph data models add labels to the nodes or edges or both. Other graph data models include GOOD [23], G+/GraphLog [19,20], and Hyperlog [22].

4.3 Our Solution

Our graph data model is based on feature structures [24], ψ -types [25], conceptual graphs [26], and terminological description logics [27] which are representation formalisms used in computational linguistics. The representations developed for capturing natural language are typically the most expressive, and they are flexible enough to represent a variety of domains.

Our graph data model can be defined in a fashion similar to the relational model. A relation is defined mathematically as a set of n-tuples. Graphs are defined in graph theory as a collection of vertices and edges $G = (V, E)$. The vertices in a graph refer to genomic objects or n-ary relations and the edges refer to binary relationships (links) between them. Relational databases are often accessed using the declarative language SQL. We have developed a declarative language called WEB to access a graph database [18,28].

There are four data types in this graph data model: vertices, edges, labels, graphs. A graph is a collection of vertices, edges and labels where:

- Vertices are nodes in the graph which model the simple concepts and n-ary relations of the domain.
- Symbols are nodes in the graph which identify external objects.
- Edges connect two vertices. There cannot be multiple edges between two vertices with the same label.
- Labels on the edges define the type of relation which holds between the two vertices.

Thus, an edge is a relation between two vertices and one label.

Operators are needed for defining data in a database, updating a database, and querying a database. In a graph data model, these operators access and manipulate graphs. The basic model presented here requires only seven operators:

1. Create a new, empty graph.
2. Add a new vertex to a graph.
3. Add a new label with a specific name to a graph.
4. Add a new symbol to a graph with a given name.
5. Add a new edge with a given label, source node and destination node to a graph.
6. Retrieve a label from a graph given its name.
7. Retrieve an edge from a graph given some (or all) of its components -- label, source vertex, edge vertex. Thus, edge retrieval is actually a family of $2^3 = 8$ operators.

For simple applications these operators are sufficient. A more complete graph data model would include more operators on graphs, such as those discussed in [28, 29].

There are two database-independent integrity constraints on this graph data model:

1. Labels in a graph are uniquely named.
2. Edges are composed of the labels and vertices of the graph in which the edge occurs, i.e., $G=(V,E,L)$ where $E \subseteq L \times V \times V$.

These constraints correspond roughly to the integrity constraints for the relational data model: (1) No component of the primary key of a base relation is allowed to accept nulls; (2) The database must not contain any unmatched foreign key values.

5. Graph Database Management Systems

Graphs simplify the design of relational and object-oriented databases for genomics. Instead of forcing the biologist to use the language of database systems, graphs provide a language which a biologist is more comfortable using to describe the science. Graphs are also familiar to computer scientists and are precise enough that they can be translated into database structures without loss of information.

5.1 Requirements

Database systems consist of a collection of data items (a database) described using a data model and stored in a database management system. The software for database system consists of minimally a database and a database management system (DBMS) and may also contain other software, such as design tools, utilities, application environments, or report generators.

When a new data model is defined, a useful possibility to evaluate is the development of a new DBMS which stores the data directly. The approach may lead to a specialized domain-specific DBMS or a new, generally useful, type of DBMS. This strategy has been used repeatedly in database research leading to the development of relational, object-oriented, semantic, logic and graph database management systems.

5.2 Available Technologies

Two approaches to developing a new database system for a new data model are: (1) to develop a process to translate each schema individually from the new data model to an existing one and (2) to develop a frontend to an existing DBMS which accepts data described using the new model. The first approach is frequently used to develop relational databases from an entity-relation diagram [30]; within genome informatics it has been used to develop OP/M [31] which captures objects and protocols of biological experiments using a relational database. The second approach has been used to develop semantic databases as frontends to relational databases; within genome informatics LabBase [4] has been developed to model biological experiments with an object-oriented DBMS backend, and Crystal [32] is an active DBMS which supports physical mapping using an object-oriented DBMS.

The strategy we have chosen to apply a graph data model to genome research is to develop a new frontend to an existing DBMS.

5.3 Our Solution

A graph database system must provide not only a good representation for describing genome data but also data structures and operations for storing, manipulating, and retrieving genome data from a database. There are three implementations of our graph database system which incorporate the functionality described here. The first is implemented as 4,000 lines of Allegro Common Lisp [18]. The second implementation is designed to provide a robust foundation for multiple large databases, including the recovery and security features of a commercial object-oriented DBMS. The third implementation is a restricted version designed to be manageable for sites with limited resources.

A common method for describing the architecture of a database management system is in terms of three layers: external, conceptual and internal [33, 34]. These layers are built to support and provide necessary functionality for application development, data querying and manipulation, and data storage, respectively. The external layer of a graph database management system provides interactive views on the data stored in the database. Applications which interact with the database use the interactive views to access data as necessary. The conceptual layer provides graph operations. In the conceptual layer, graphs have their own existence and are treated as entities which can be manipulated and stored. The internal layer is the set of data structures which store and access the data as graphs, vertices, edges, and edge labels. The architecture is diagrammed in Fig. 3.

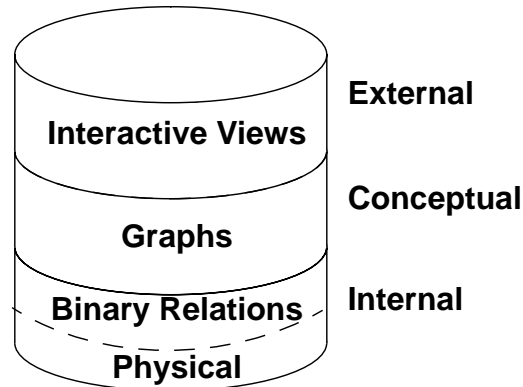


Figure 3: Graph DBMS Architecture

The internal layer is a database management system for binary relationships, providing data storage and retrieval capabilities to support the simple graph data model described in the previous section. An application which directly accesses the internal layer has also been built. The database management system corresponding to the internal layer stores and retrieves the vertices, edges, and edge labels of a graph. These graphs correspond to collections binary relations in mathematical logic. Because it is functionally complete as a simple DBMS, we have named it a binary relation database management system (BRDBMS). BRDBMS was also built as a layered architecture which separates the binary relationship operations from the physical storage mechanism. Modularity was emphasized because we realized that the storage mechanism would change as development progressed. Each module provided a set of functions which were relatively independent. The physical layer handles storage and access to data in a persistent data store. The physical layer provides data structures to store and retrieve graphs, and their vertices, edges, and edge labels, but it does not provide operations to access the data associated with those graph objects, such as the name of an edge label. Those operations are provided in the binary relation layer. The binary relation layer also provides mechanisms for treating a graph as a database and querying against it.

The conceptual layer provides more extensive query capabilities. The internal layer provides the ability to retrieve edges, and the conceptual layer augments that behavior by combining the simple edge queries into query graphs of arbitrary complexity. An example query graph is given in Fig. 4 which asks:

What genes on human chromosome 4p11 are expressed in the brain and have known mouse homologs?

The query interface shown is part of the third implementation.

The external layer provides support for user views that allow full access to data (not just read-only). Storing all the data in a single graph data structure is too unwieldy for large databases. We

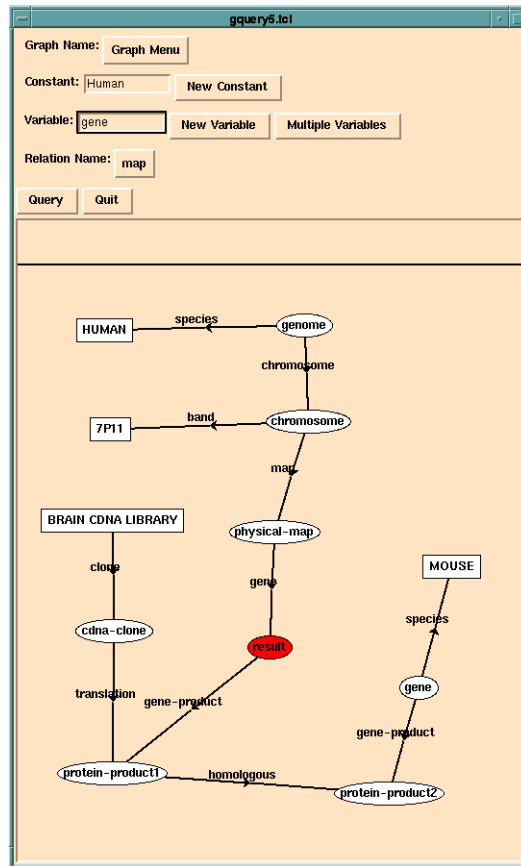


Figure 4: A query tool displaying the query “What genes on human chromosome 7p11 are expressed in brain and have known mouse homologs?”

have investigated in detail decomposing graphs using a variety of technologies, including abstract data types, constructive type theory [18], and object-oriented programming techniques [35]. Each of these has their own benefits, and a combination of these technologies provides interactive views to the DBMS. The interactive user views provide the functionality of a third generation database management system [36].

Our graph DBMS architecture has been implemented in three systems as mentioned above. The Allegro Common Lisp implementation supports the functionality required for all three layers, but the second and third implementations are incomplete.

The second system is a complete implementation of BRDBMS, but the conceptual and external layers have not yet been implemented. BRDBMS is implemented in Parc Place Smalltalk and consists of 50 classes (approximately 5000 lines of code). BRDBMS was developed within a commercial object-oriented database (Servio’s Gemstone) which provides features for security, recovery, and transaction management. Data may be stored either in Gemstone or in the Smalltalk image. A data entry tool has also been built on top of it to support a high-volume laboratory at Baylor College of Medicine.

The third system is implemented using Unix C-Shell scripts and stores data in Unix files. The internal layer has been completed and most of the functionality of the conceptual and external layers has been implemented. This implementation is designed to provide less functionality than the other system, but it does not require the resources of a large DBMS. It is a little slower and not as powerful as the other implementations, but still provides some security features, transaction log-

ging, and recovery mechanisms. BRDBMS is implemented as a 350-line Unix C-Shell script, and the other two levels have been developed using C-Shell scripts and logic programs. Databases in excess of 1,000,000 binary relations can be created with no noticeable degradation in performance. This implementation is currently the database behind a system to capture Human X chromosome mapping data which is being used in the Baylor College of Medicine Genome Center.

5.4 Application to Multidatabase Systems

Because data is currently divided over multiple databases at several sites, a graph database system must connect to other database systems. Robbins [15] discusses an approach which restricts the operations of the databases to simplify the transfer of data. One approach to transferring data between databases is to view a multidatabase system as a graph.

In a graph database system, the storage mechanisms is a graph, the database is a graph, and a multi-database system is a graph. To support the ability to combine graphs across databases, each graph object -- vertex, edge, label, graph -- has a unique identifier based on its storage location. This unique identifier is sometimes called a “handle” in computer science or an “accession number” in genome databases. Our handles consist of four parts:

1. Site number -- the university or other institution.
2. Database number -- the database within the site.
3. Storage graph number -- the storage graph which contains the data.
4. Graph object number -- the id within the storage graph.

This approach is not limited to storage in a graph database nor does it require that the databases have the same data model. The storage graph number may be ignored for an object database (or refer to a segment or version) and for a relational database it can refer to a mapping associated with each table that will map the object number to a specific key. Each site can be responsible for connecting external users to the appropriate database and retrieving the appropriate data. The site numbers can either be stored in a globally accessible table or derived from another source, such as Internet ids.

6. Discussion

Database systems have been undergoing development for over thirty years from a steady progression of new technologies from other areas of computer science. Representing genome data is a difficult task, but database research has developed paradigms which are useful for the sequencing and mapping needs of the Human Genome Project. The network and hierarchical databases were developed in the 1960s and prevalent in the 1970s. Relational database theory was developed in the late 1960s, implemented in working systems in the 1970s, and started to reach maturity in the 1980s [37]. Object-oriented databases are based on the ideas of the 1970s, developed in the 1980s and are now robust enough for many real-world applications [38,39]. Graph-based data models are based on frameworks set in place in the 1980s to model complex structures and to take advantage of improvements in human-computer interaction using graphical workstations [40] and are starting to be used in working systems.

First generation database systems provided data persistence and languages for manipulating the data. They included the network and hierarchical databases. The network database stores data as

binary, many-to-one relationships. The first generation data models may be considered to be physical data models.

Second generation database systems provide data independence and non-procedural data manipulation languages. A level of abstraction is added above the physical storage. Users manipulate the data using this level of abstraction and are isolated from the intricacies of the specific storage mechanism. Relational databases, most early object oriented-databases and many semantic data models are included in second generation systems. Semantic data models originated from semantic networks and are used in genetics as ER diagrams [30] and ACeDB [5]. Semantic data models often have a graph-like structure built from functional arcs or a small set of type constructors.

Third generation database systems provide extensibility. They allow the creation of new data types, constraints and rules on the data types, and high-level access languages. Another level of abstraction is added to the second generation systems where the user may extend the database in several ways using the functionality of the second-generation level of abstraction. For example, the user must be able to use types other than just records or objects, such as arrays, sequence, sets, and functions. These systems have been under development for several years and are beginning to be commercially available.

There is a strong parallel between data model development and programming language development. In first generation database systems, the manipulation of data is closely tied to how it was stored, much like machine language programs are tied to the details of the processor. In second generation database systems, a level of abstraction separates the user from the storage mechanisms, much like assembly language programs add a level of abstraction between the programmer and the processing chip. In third generation systems, the user can define new constructs at the previous level of abstraction and manipulate the data using a new, higher-level language, much like commonly used programming languages such as C, Fortran, PL/1, COBOL, and Pascal allow the user to extend processing via user-defined procedures, functions, and subroutines.

Graph database management systems combine functionality of relational and object-oriented systems. Graphs provide more flexibility and extensibility than relational systems and more support for relationships than object-oriented systems. Since genome data is graph-like, a major advantage of a graph DBMS is the ability to store the data in a graph structure without any coercion of the data.

Although the description of the data model and architecture are simplified in this paper, they do demonstrate two of the results we have found by developing a practical application which is supported by a theoretical foundation. First, a layered database architecture seems to be the most appropriate for developing data-intensive genome applications, though this may not be true in areas with more complex transactions, such as computer-aided design and manufacturing (CAD/CAM). The second result is that a graph representation is appropriate for laboratory databases. Because genome research is changing rapidly, the flexibility of a graph representation is useful.

Graph database management systems are a new technology. There are no commercially available systems which is a disadvantage in comparison to relational or object-oriented database systems. We believe that the advantages of graph databases for genome research outweigh the uncertainty of being an unproven technology.

7. Conclusion

We have investigated using graphs as the foundation for database systems by developing a DBMS based on graphs. We have demonstrated that:

1. Graph-based representations are useful for representing genome data.
2. A graph data model tailored to the requirements of genome data can be used as the basis of a database management system.
3. A graph database management system is a viable technology for storing genome data.

We have found that graph-based technologies are useful in multiple aspects of database development and plan to expand on the capabilities of graphs for developing genome databases.

8. Acknowledgments

The authors thank Andy Arenson, Bob Cottingham, Pamela Culpepper, Dan Davison, Joanna Power, Randy Smith, and Wayne Parrott for frequent discussions of the ideas in this paper. This work was supported by the W.M. Keck Center for Computational Biology, the Baylor Human Genome Center funded by the NIH National Center for Human Genome Research, a grant to C.B.L. from the Department of Energy, and fellowships to M.G. from the National Library of Medicine and from the Department of Energy.

References

1. Connell J, Shafer LB. *Structured Rapid Prototyping*. Yourdon Press (Prentice Hall), Englewood, NJ, 1989.
2. Prieto-Diaz R, Arango G, eds. *Domain Analysis and Software Systems Modeling*. IEEE Press, 1991.
3. Coleman D, Arnold P, Bodoff S, Dollin C, Gilchrist H, Hayes F, and Jeremaes P. *Object-Oriented Development the Fusion Method*. Prentice Hall, Englewood, NJ, 1994.
4. Goodman N, Rozen S, and Stein L. LabBase: A Database to Manage Laboratory Data in a Large-Scale Genome-Mapping Project. *IEEE Engineering in Medicine and Biology*. In this issue, 1995.
5. Durbin R, Thierry-Mieg J. A C. elegans database. Available via anonymous ftp from lirmm.lirmm.fr, cele.mrc-lmb.cam.ac.uk and ncbi.nlm.nih.gov 1991.
6. Cinkosky MJ, Fickett JW, Keen GM. A New Design for the Genome Sequence Data Base. *IEEE Engineering in Medicine and Biology*. In this issue 1995.
7. Lehmann F. *Semantic Networks in Artificial Intelligence*. Pergamon Press, Oxford, 1992
8. Sowa J. *Principles of Semantic Networks*. Morgan Kaufmann, San Francisco, CA, 1991.
9. Thieffry D, Thomas R. Logical Analysis of Genetic Regulatory Networks. In H. Lim, ed., *Proceedings of The Third International Conference on Bioinformatics and Genome Research*. World Scientific Publishing. June, 1994.
10. Ochs R. A relational database model for metabolic information. In H. Lim, ed., *Proceedings of The Third International Conference on Bioinformatics and Genome Research*. World Scientific Publishing. June, 1994.
11. Hofstaedt R. Modeling and Visualization of Metabolic Bottlenecks. In H. Lim, ed., *Proceedings of The Third International Conference on Bioinformatics and Genome Research*. World Scientific Publishing. June, 1994.

12. Karp P, Paley S. Automated drawing of metabolic pathways. In H. Lim, ed., *Proceedings of The Third International Conference on Bioinformatics and Genome Research*. World Scientific Publishing. June, 1994.
13. Graves M. Integrating order and distance relationships from heterogeneous maps. In L. Hunter, D. Searls and J. Shavlik, eds., *Proceedings of the First International Conference on Intelligent Systems for Molecular Biology (ISMB-93)*, AAAI/MIT Press, Menlo Park, CA, July 1993.
14. Mirkin BG, Rodin SN. *Graphs and Genes*, volume 11 of *Biomathematics*. Springer-Verlag, 1984.
15. Robbins B. An Information Infrastructure for the Human Genome Project. *IEEE Engineering in Medicine and Biology*. In this issue 1995.
16. Codd EF. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6): 377-387, June 1970.
17. Codd EF. Data models in database management. In M. Brodie and S.N. Ziles, eds, *Proc. Workshop on Data Abstraction, Databases, and Conceptual Modelling*, June 1980. Also, *ACM SIGMOD Record* 11(2).
18. Graves M. *Theories and tools for designing application-specific knowledge base data models*. Ph.D. Thesis, University of Michigan, Dept. of Electrical Engineering and Computer Science. University Microfilms, Inc. April, 1993.
19. Consens MP, Mendelzon AO. *GraphLog: A Visual Formalism for Real Life Recursion*. In H Garcia-Molina and HV Jagadish, editors, *Proc of the 1990 ACM SIGMOD International Conference on Principles of Database Systems*, page 388, May 1990.
20. Consens M, Mendelzon A. *The G+/GraphLog visual query system*. In H Garcia-Molina and HV Jagadish, editors, *Proc of the 1990 ACM SIGMOD International Conference on Management of Data*, page 388, May 1990.
21. Gyssens M, Paradaens J, and D Van Gucht. *A graph-oriented object model for database end-user interfaces*. In *Proceedings of 1990 ACM SIGMOD Conference on Management of Data*, 1990.
22. Levene M, Poulouvasilis A. *The hypernode model and its associated query language*. In *Proc Fifth Jerusalem Conference on Information Technology*, pages 520--530, 1990.
23. Gyssens M, Paradaens J, and D Van Gucht. *A graph-oriented object database model*. In *Proceedings of ACM Conference on Principles of Database Systems*, 1990.
24. Kasper RT, Rounds WC. A logical semantics for feature structures. In *Proceedings of the 24th Annual Conference of the Association for Computational Linguistics*, pages 235-242, 1986.
25. Ait-Kaci H. *A Lattice-Theoretic Approach to Computation Based on a Calculus of Partially-Ordered Type Structures*. Ph.D. Thesis, Computer and Information Science, University of Pennsylvania, Philadelphia, PA, 1984.
26. Sowa J. *Conceptual Graphs*. Addison-Wesley, Reading, MA, 1984.
27. Nebel B, Smolka G. Representation and reasoning with attributive descriptions. In K.H. Blasius, U. Hedstuck and C.R. Rollinger, editors *Sorts and Types in Artificial Intelligence*, volume 418 of *LNAI*, pages 112-139. Springer-Verlag, 1990

28. Graves M, Bergeman E, Lawrence CB. Querying a Genome Database Using Graphs. In H. Lim, ed., *Proceedings of The Third International Conference on Bioinformatics and Genome Research*. World Scientific Publishing. June, 1994.
29. Graves M, Bergeman E, Lawrence CB. A Graph-Theoretic Data Model for Genome Mapping Databases. In the Proceedings of the Hawaii International Conference on System Sciences-28, Biotechnology Computing Track. January, 1995.
30. Chen PP. The entity-relationship model: toward a unified view of data, *ACM Transactions on Database Systems* 1:1, pp. 9-36, 1976.
31. Chen IA, Markowitz VM. An overview of the Object Protocol Model (OPM) and OPM Data Management Tools, TR LBL-33706, Lawrence Berkeley Laboratory, 1994.
32. Lee AJ, Rundensteiner E, Thomas S, Lafortune S. An information model for genome map representation and assembly. *ACM 2nd Int. Conf. on Information and Knowledge Management*. Nov, 1993.
33. Tsichritzis DC, Klug A, eds. The ANSI/X3/SPARC DBMS Framework: Report of the Study Group on Data Base Management Systems. *Information Systems* 3 (1978).
34. Date CJ. *An Introduction to Database Systems Volume 1, 5th Ed.* Addison-Wesley, 1990.
35. Bergeman ER, Graves M, Lawrence CB. "Viewing Genome Data as Objects for Application Development". Proceedings, Third International Conference on Intelligent Systems for Molecular Biology (ISMB-95). AAAI Press. July, 1995.
36. The Committee for Advanced DBMS Function. Third-Generation Database System Manifesto. *SIGMOD Record*, 19(3):31-44, 1990.
37. Ullman J. *Principles of database systems*. Computer Science Press, Rockville, MD, 2nd edition, 1982.
38. Zdonik SB, Maier D, eds. *Readings in object-oriented database systems*. Morgan Kaufmann, San Mateo, CA, 1990.
39. Cattell R. *Object Data Management*. Addison-Wesley, 1991.
40. Ioannidis Y. Special issue on advanced user interfaces for database systems. *ACM SIGMOD Record*, 21(1), March 1992.