To appear in the Proceedings of the Hawaii International Conference on System Sciences-28, Biotechnology Computing Track. January, 1995.

A Graph-Theoretic Data Model for Genome Mapping Databases

Mark Graves,[1] Ellen R. Bergeman,[2] Charles B. Lawrence[1]

[1]Department of Cell Biology & [2]Department of Human and Molecular Genetics,
Baylor College of Medicine, One Baylor Plaza, Houston, TX 77030
email: {mgraves,erb,chas}@bcm.tmc.edu

Abstract

Graphs are a natural foundation for genome map databases. Mapping and other genomic data can be clearly represented by graphs, and graphs can be stored in a database. Graphs are defined as a collection of nodes and arcs and can represent genomic objects and relationships between them. Mapping databases are needed to store the rapidly growing amount of mapping data. These databases must store the information contained in both published maps and laboratory notebooks. We describe a graph database which can store mapping data directly as graphs and formalize it as a graph-theoretic data model.

1. Introduction

Graphs are a good way of representing genomic data. Graphs have been used to describe mapping relationships [13,4], gene regulation [32], metabolic pathways [30,18,22] and phylogenetic trees [28]. Graphs are an especially good representation for mapping information and can be used to describe the concepts involved in laboratory development of maps and in helping to model the inconsistencies and uncertainties which are contained in published maps.

Genome mapping databases are needed. Mapping data is being generated at an astounding rate (CEPH, MIT). The Human Genome Project (and other similar projects) are spending considerable resources to develop maps of the genome. CEPH has developed a framework map with 3500 markers, including a map with 2000 markers developed by the Genethon project. In addition, these maps must be integrated with maps generated by other labs, with the 40,000 loci in the Genome Data Base, and the various libraries of YACs, cosmids, cDNAs, and sequences being generated by rapidly expanding laboratories. These data sources must be integrated so that they share overlapping information.

Graph databases are a good way of storing and querying genome map data. Graphs can describe mapping information and can connect that information to descriptions of other genomic data. Rather than translating a description of a map into relations or objects, a map can be described directly as a graph. This graph description can be stored directly in a database and queries of the database can retrieve the mapping data. One advantage of storing maps as graphs is that it is easier for the user to follow the structure of the database because the structure is analogous to the familiar structure of maps.

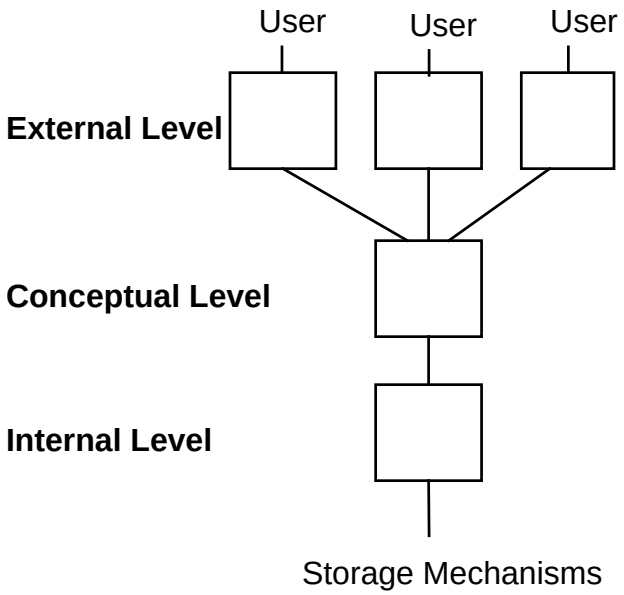
Formalizing graphs as a graph-theoretic data model defines a useful database. A data model is a formal description of the data structure(s) used in a database to represent data and the operations on the data structure. Describing the database at a formal, abstract level clarifies how data is to be represented and how it can be manipulated. A data model isolates the description of the database from the details of how the specification can be met. For example, a relational database is defined by the relational data model [6]. Data in a relational database is to be represented as relations. The data model also defines operations on relations which specify how to combine relations and retrieve data from them. A graph-theoretic data model is based on the definition in graph theory that a graph is a set of vertices and edges (nodes and arcs). Data in a graph-theoretic data model is to be represented as a graph. Operations on graphs must also be defined.

We present the architecture and data model of a graph database along with a description of its use for genome mapping. Database architecture is often described as a layered architecture. We define the architecture of a graph database and describe its three layers: how the user interacts with the database, how graphs are represented and manipulated by the database, and how graphs are stored and retrieved. The middle layer is formalized by a graph-theoretic data model. We define our graph-theoretic data model and describe how graphs are represented and ma-

nipulated. Finally, we show how the graph database can store and retrieve mapping information.

2. Graph Database

A database consists of three architectural levels: external level, conceptual level, and the internal level [34,10]. The external level describes how users interact with the database management system. Users enter data and query the database using views, which are simplified versions of the database based on the needs of the user. The collection of user views defines the external level. The conceptual level describes the overall structure of the database including the mechanisms used to organize data. The conceptual level ties the pieces of the database together and specifies the data structures used to organize data in the database and is formalized by a data model. The internal level describes how the physical storage devices are used to store data. The internal level specifies the interface between the conceptual level and the physical storage mechanisms available on the computer, as shown below.



We describe a graph database by defining it at the conceptual, internal, and external levels: data modelling, data storage, and user views.

2.1 Data Models

A data model is a formalization of the conceptual level. It defines the types of data the database stores and the operators which manipulate them. Data models were first proposed by Codd [6] to describe the relational database in an abstract, formal manner. A data model describes and restricts how data is to be represented and manipulated.

No data model is good for all domains. Data models which are useful for modeling financial records or inventories will differ from data models which are useful for modeling the behavior of design within computer aided design applications or genomic applications. A data model is a formal means of representing and manipulating the structures of a database and is usually defined in terms of type constructors, operators, and integrity constraints [9,37,2]. Common data models include the relational data model for relational databases and F-logic [24] or complex objects [20] for object-oriented databases [38]. Other data models include the network, functional (or semantic) and logic data models. The network data model is a physical data model (like hierarchical). It stores data as binary, many-to-one relationships, but is tied to the storage mechanism used. Logic data models are defined using mathematical logic and often use a subset of first-order predicate logic. Semantic data models originated from semantic networks that were developed as tools in artificial intelligence for representing the semantics of natural language. They often have a graph-like structure built from functional arcs or a small set of type constructors. Although graph-theoretic data models appear most similar to semantic data models, they may be defined as a logic [12,7].

Genome mapping data is built primarily from relationships, but the relational data model is too restrictive in its ability to be extended. Object-oriented databases are extensible but emphasize the behavior of objects and insist that each object have its own identity. This is of questionable utility in modeling “mass nouns” such as colonies and is distracting when representing abstract genomic concepts such as experiments, hybridizations or locations on a map. Semantic data models are better than object-oriented ones but are still too object-centered for good models of the relationships in mapping databases. Logic data models have traditionally been useful for extending relations, and we have incorporated some functionality of logic databases in our system. Graph-theoretic data models are not as well known but are the best data models currently available for modeling mapping data. Robust, commercial database systems are not available for semantic, logic, or graph-theoretic data models, so we are developing a graph-theoretic database management system on top of a commercial object-oriented one.

Graph-theoretic data models formalize the representation and manipulation of graph data structures for database systems by defining a flexible collection of graph-oriented type constructors and operators which create and access graph data structures. All graph-theoretic data models have as their foundation the mathematical definition of a graph, which is a collection of nodes and edges. The data models are usually used to support an application, such as

visual querying [7], end-user interfaces [17], hypertext systems [27], or extensible databases [12].

A graph database is defined at the conceptual level by a graph-theoretic data model that formalizes the representation of the data structures stored in a database as a graph. Unlike the relational data model which has one incarnation, there are several extensions to the definition of graphs which form foundations for different graph-theoretic data models. Most graph-theoretic data models usually add labels to the nodes or edges or both. Sometimes the basic definition of a graph is changed by allowing nodes to encapsulate graphs [27] and allowing edges to relate to other edges [12]. Related data models include GOOD [16], G+/GraphLog [7], and Hyperlog [27]. Our graph-theoretic data model extends the definition of a graph to include features such as: labeling edges with types, allowing higher-order relations, and packaging other graphs as an encapsulated vertex. The added features make our graph-theoretic data model more useful than other graph-theoretic data models for modeling genomic data.

Operators are defined in a graph-theoretic data model which manipulate the graph. Most graph-theoretic data models have operators which add and delete nodes and edges in the graph. Each data model usually adds other operators, such as an abstraction mechanism to group nodes [16]. We have developed a graph-theoretic data model which is geared toward representing genomic data and includes an operator for querying the database which is described below. We have also developed a database programming language called WEB which implements some of our operators and can be used to build and query a graph database [12]. WEB is a declarative programming language like SQL, but is based on a graph logic rather than relational operators. WEB can be used directly, embedded in another programming language, or accessed through a graphical user interface.

We give a more detailed description of our representation of graphs and the operators which manipulate them in sections 3 and 4.

2.2 Data Storage

A *binary relationship* links two objects through a relation. The internal level of the graph database is defined by specifying how the binary relationships that make up graphs are stored and retrieved. We have developed efficient data structures for storing and indexing binary relationships in a database. By creating index tables for all the needed binary relationship queries, complex queries can be answered more efficiently. Each binary relationship query consists of an index table lookup.

The physical storage of binary relationships can be handled either by developing our own physical storage scheme or borrowing one from another database. We have developed a file-oriented storage mechanism which we currently use and are in the process of developing storage mechanisms which use a commercial database system (Gemstone) for data security and recovery.

2.3 User Views

Graphs are the external view of the database. Each view of the database correspond to the needs of different users or task. No user wants to see all graphs for all genomic data, and no user wants to sort through a lot of extraneous information to find the few facts of interest. A scientific database must model multiple views of the data. Views correspond to different users, different tasks, or different scientific theories. Graphs facilitate the definition of external views because any collection of binary relations can be selected to form a graph. The meaningful graphs correspond to concepts in the domain.

The external view describes all interactions between the user and the database. The external view of a graph database specifies the graphs which a user can use in creating, manipulating, and querying data. Graphs have been used to specify schemas at this level using Entity-Relation diagrams [5]. We also use graphs to specify the queries. Querying in a database is done using a query language which implements some of the operators of the conceptual level. Query languages are also used to create user views. In a relational database, this is often SQL. SQL is a query language that implements some of the relational operators and which the end user can either use directly or embed in a programming language to access the database. Application developers use SQL to define external or user views which allow access to the part of the database which is relevant to the application. We use our graph logic programming language WEB for querying and data entry.

3. Representation

A relation is defined mathematically as a set of n-tuples. Graphs are defined in graph theory as a collection of vertices and edges $G = (V, E)$ [33]. The vertices in a graph refer to genomic objects or n-ary relations and the edges refer to binary relationships (roles) between them. We have extended the basic theoretical definition of a graph to capture complexities which occur in genome data.

The type of relationship between two genomic objects is also important, thus we add a collection of edge types to the definition of a graph. The edge types specify the relation between two vertices or the role that one of them has with respect to the other. For example, valid relations between a cosmid and a YAC would include “hybridizesTo”, “generatedFrom”, or “sharesSTS”, or a plasmid might be in relationship to sequence or location objects. The edge

types are equivalent to the relations in a relational database restricted to two attributes.

Another extension is an abstraction mechanism for graphs which defines encapsulated graphs as vertices. These packaged vertices can be related to other vertices in a graph-like manner. This mechanism is useful for representing reagent libraries or collected experimental results. The definition of a graph as a collection of vertices and edges is extended in our data model with relation types which label the edges and an abstraction mechanism which defines other graphs as new vertices in the current graph. Thus, a graph is a collection of:

- Simple vertices which are the nodes of the graph and model the simple concepts and n-ary relations of the domain.
- Edges which connect two vertices. There can be multiple edges between two vertices (with the same or different relation types).
- Relation types which label edges. Thus, an edge is a relation between two vertices and one relation type.
- Graphs which are packaged into vertices. Each packaged graph also defines simple vertices, edges, relation types, and other graphs packaged into vertices.

Another feature included in the graphs is relations between relations (higher-order relations). For example, stating that one ordering in a genetic linkage or radiation hybrid map is more likely to occur than an alternative ordering [13]. This extension actually simplifies the database design and as well as increases the usefulness of the graph database for modeling genomic data. To do this, we let relation types and edges be treated as vertices. This leads to the design:

1. There are four kinds of vertices: simple vertices, packaged graphs, relation types, and edges.
2. Edges connect two vertices and are labeled with a relation type.
3. Graphs occur only within the context of packaged graphs. The database consists of one packaged graph which is the root of the packaged graph hierarchy.

Our design for the graph data structure consists of a directed, labeled, possibly cyclic graph which maintains hierarchically-ordered graphs. It also maintains all positive relationships, inverse relationships, and uses of each relationship in an access-efficient data structure. The data structure allows for querying by partial specification of any graph, and the graph querying algorithm works by returning all most general graphs in the database which are more specific than the query. Future representation features include computed arcs whose destination is calculated for each query and which depends upon the current state of the database.

4. Operations

Operators are needed for defining data in a database, updating a database, and querying a database. Operators, such as “select”, “project”, and “join” are defined for a relational database and specify how relations can be accessed and manipulated. We must also define operators for accessing and manipulating graphs. Because updating operators can be defined in terms of building and retrieving graphs, we will describe operators which create graphs in a database and define algorithms which query graphs in a graph database.

There are four main operators on graphs in our graph-theoretic data model. These operators build graphs in the database, query against the database or a packaged graph within it, and delete graphs which are no longer needed. In the future, additional operations, such as inference rules, may be added. The operators are:

- enter data using a graph
- query a graph against the database
- query a graph against a packaged graph
- delete a subgraph from the database

Querying takes place against either a packaged graph or the entire database. The query operator against the database uses the packaged graph query operator to query each graph, then combines the result. We will discuss only building the graph and querying packaged graphs here.

4.1 Build

Traditionally, databases have been developed which had a particular structure, domain concepts and relationships were forced into that structure, and the user was forced to learn the organization of the database to ask queries. We have started with the concepts and relationships in the domain, discovered a data structure which can model those concepts and relationships, and allow the user to ask queries where the structure of the query follows closely the organization of information in the domain. This approach allows a user to build a database which has a structure analogous to the structure of the domain.

A graph database is built using graphs defined by the external view. To insure integrity of data, the user is not allowed to create arbitrary nodes and arcs, but all data entry is done through a user view. The user views for creating data are logic programs, written in WEB, which will create appropriate binary relationships when executed.

4.2 Querying

The graph query algorithm decomposes the query graph into binary relationships and queries storage for matching binary relationships. A binary relationship query is the mechanism for retrieving data from storage. Because

all graph data is stored as binary relationships, all graph query algorithms must retrieve data as binary relationships. The binary relationships from storage are combined into graphs as specified by the query graph. These resulting graphs are the result of the graph query.

A query is evaluated by a graph query operator which:

1. translates the graph query into a series of binary relationship queries,
2. queries the stored data for binary relationship which match,
3. builds the graph query result using the binary relationship query results, and
4. returns the graph query result.

Querying complex graphs against a large database can be prohibitive. A useful database query engine must be powerful and flexible to allow a variety of queries. In addition, it must be very efficient to retrieve those queries as quickly as possible. Unfortunately, those are conflicting demands and a trade-off must be made between expressiveness and speed.

Rather than compromise between the two approaches, we are developing two separate query engines. The first is an efficient query engine that allows for a useful subset of commonly asked queries, which are implied by the structure of the genome maps. The second is a more powerful query engine that allows for the specification of a theoretically complete set of queries and is based on the structure of the graph-theoretic data model. The first query engine will be updated as users ask more complex queries and as the types of genome maps evolve. Initially, we are developing a form-based query interface for the first query engine and a graphical user interface for the second query engine. Later, we may extend the graphical user interface to call the more efficient query engine when possible.

The first query engine restricts the form of the query graph to be rooted, directed, acyclic graphs (sometimes called a rational tree). The algorithm is based on path following extended to allow for equality constraints on the paths [23]. The user specifies paths from a root vertex to the "leaves" along with a set of constraints on them. The algorithm works by following the paths specified by the user in the query and returning the values from the database which correspond to the end of the paths. Equality constraints on the paths require that two paths must point to the same object. This algorithm is related to algorithms currently being used for natural language processing systems [35] and knowledge representation and reasoning systems [29] but modified to query against a database. The algorithm is described in more detail in [14].

The second query engine defines queries using the same operations as are used to define the data model. Thus, it is complete in the sense that it can retrieve any graph which is directly stored in the database. The user defines a query by defining a partially specified graph. The algorithm compares the query graph to the database using efficient access mechanisms and then returning the graphs in the database which "match" the query. Mathematically, the algorithm returns the most general graphs in the database which are more specific than the query graph. The query algorithm works by decomposing a graph into a set of binary relationships. Each binary relationship is queried against the database and the results of the binary relationship queries are combined into the graph query results. The implemented algorithm is described in more detail in [12].

5. Genome Mapping Databases

Preliminary analysis of complex genome map information reveals that the complexity in genome maps can be effectively represented using graph-based abstractions. Graph-theoretic data models are useful for representing map information, such as marker definitions, distance and order relationships, alternative splicing, allelic variation between populations, gene regulation mechanisms and representation of homologies.

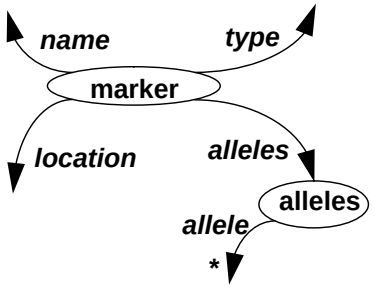
There are two types of genome mapping databases which are needed. The first is a laboratory database which captures the data currently stored in lab notebooks. The second is a database which captures the relationships presented by published maps. The laboratory database must store: markers, probes, clones, primer pairs, PCR conditions, allele information, etc. The published database must be capable of capturing a variety of map related data which includes, but is not limited to: sequence, physical, genetic linkage, radiation hybrid, cytogenetic, and gene maps. We have shown in [13] how published distance and order relationships can be represented as graphs (though not in a graph-theoretic data model or a graph database) and here we show how a laboratory database can be defined using graphs.

5.1 Laboratory Databases

There are many types of markers that are useful in finding the location of genes. One useful type is the *dinucleotide repeat marker.* At many locations in the genome, two nucleotide bases, such as CA, are repeated. The number of repeats can vary between people and the pattern of inheritance of these different *alleles* can be examined within a family to estimate how likely a marker is to be inherited with a gene being searched for. These probabilities can be used to estimate the location of the gene with respect to other markers whose locations are known by converting the probability to a distance using a mapping function [31].

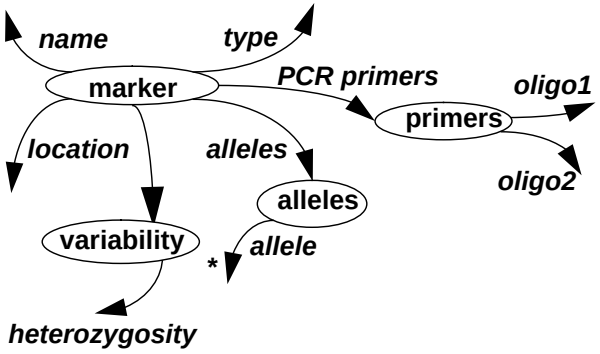
There are various factors which make some dinucleotide repeat markers more useful than other ones, and discovering useful markers is a fairly involved process which can be supported by databases modeling the appropriate laboratory data. One domain model for representing the relations between markers and the clones which are used to discover them is presented in Figure 1.

Figure 1 shows the relationship of a PCR marker to a clone which contains it. The clone is related to the marker through the oligonucleotide which occurs in the sequence of the clone and which serves as the PCR primers for the marker. A clone may have many oligonucleotides. A marker is defined by a name, a location on a chromosome, a marker type, and a set of alleles.



We have chosen to represent the set of alleles as a separate graph node because of the complexity of information which may be associated with an allele set (especially when multiple populations are being studied). This also illustrates a difference between the structure which can be modeled as graphs versus objects or relations. The set of alleles is an integral part of the definition of marker and should not be separated at the conceptual level. Graph-theoretic data models allow concepts to be associated with meaningful subgraphs while other approaches would require that a separate entity be formed for the allele set.

Other information is also important for certain types of markers or particular applications. For example, a dinucleotide repeat marker is also defined by the oligonucleotides which surround it and variability measurements of a marker such as heterozygosity are very important in choosing markers to be used for linkage analysis.



Other factors may also be important. For example, to insure that the appropriate repeat counts are read from a gel, reference sizes are necessary which correspond to commonly available sources of DNA. While not an integral part of the definition of a marker, reference sizes are required for some application. This also shows that the graph representing the structure of a concept may not always have a tree structure.

5.2 Released Databases

Mapping data must be transferred into published maps. This cannot be done automatically as human intervention is usually needed to create useful maps. However, the capability is needed to generate possible maps and connect published maps with the experimental results which support it. The graph data model supports the connection of high quality maps with the experimental data through the addition of more binary relations. For example, a physical mapping experiment may be modeled as part of a laboratory database, and the result of the experiment might be a hybridization result which would be used to support a published map. The hybridization used in the published (electronic) map would be augmented with the experimental data which supports it. We are incorporating this strategy at Baylor College of Medicine to combine cosmid-cDNA reciprocal probing results, cosmid FISH maps, and YAC-STS screenings to create maps which are annotated with experimental evidence.

Using the example of the previous section, markers can be combined into published maps. Genome maps are can be represented by graph data models by defining each map as a graph. A map is an ordering of map items where each map item is either a marker or a map [19]. A map has a local orientation [26] and may be augmented with distance relationships [13]. The ordering must be at least a partial order [15], which is sufficient to represent non-overlapping markers [13] or the endpoints of intervals [11]. Allen [1] presented 13 base relations on intervals which model order, Lee [25] showed how they can be used for inferring physical maps, and Cui [8] showed that the 13 base relations can be modeled in terms of two relations "connectedness" and "before". The foundation of connectedness is a *region*, which appears to be a good formalization of an abstract map location. This abstraction allows maps to be represented separately from the items which are being mapped.

A marker map is a graph G=(V,E,L) where V is a set of markers (or their locations), E is a set of directed edges where $E \subseteq L \times E \times E$ and L is a set of labels that are the set of order relations for the map (currently suggested to be 1, 2, 7, or 13 labels as referenced above).

6. Discussion

One of the limitations of current data base technology in developing integrated mapping databases is the inability to model the graph-like structure of maps. To store data in existing databases, the graph-like structure 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 mapping database must be able to represent the structure of maps.

Representing the data necessary for genome mapping databases is a difficult task, but database research has developed paradigms which are useful for the sequencing and mapping needs of the Human Genome Project. Relational database theory was developed in the late 1960s, implemented in working systems in the 1970s, and started to reach maturity in the 1980s [36]. 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 [38,3]. 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 [7,17,12,21]. 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. During the past few years, formalisms and prototype systems have been developed which take advantage of graph-based representations to model complex structure, but the database systems which are required for supporting an integrated mapping database are not mature enough to be commercially available. The database techniques necessary for mapping databases are still being refined by database researchers. Effective databases for integrated maps of the human genome require a model capable of representing and querying a complex, graph-like structure. Graph-based data models appear to be the most appropriate data model available for genome map databases and to be sufficient to meet the current data modeling requirements for an integrated mapping database. The idea of using graphs to model genomic information is not new [28], but the technology to model it computationally is.

Graphs can be used to model many kinds of map information, such as markers, map locations, distance and order relationships, cloning vectors, sequence, hybridization relationships, and contigs. Each of these can be complex and require different database views to describe it. Each of these views must be modeled by the database and connected to all other related genomic data. The highly inter-connected nature of genomic data also has a graph-like structure. To effectively model the graph-like structure of genome maps, it is important to define a specialized graph data model which can support the evolving nature of integrated genome maps and which allow for flexible, efficient querying on the database.

Graph data models are useful for genome databases, but the results presented here are not limited to the creation of graph databases. Graphs can be applied to many areas of genome data management. Three possible uses are as a design tool for other data access tools such as Mosaic, as a conceptual model and for exchanging data between heterogeneous databases.

There is a movement within the community to provide data access through World Wide Web and Mosaic. Hypertext links, the computational process used in WWW, are conceptually edges between nodes in a graph. The concepts of user views, map representation, and data definition within the graph-theoretic data model can be used to help in modelling and defining WWW applications. A user view which can be defined in a graph database may also be defined as a HTML (hypertext markup language) document where each meaningfull subgraph is a page in the hypertext document. This would allow for only limited data entry and no querying other than Query-By-Example, but for some applications this is sufficient. Although this technique may be used in isolation to create stand-alone Mosaic applications, this technique can also be used in conjunction with a graph database to generate HTML documents automatically.

A conceptual model is a data model which formalizes the representation and manipulation of concepts and relationship [2], and is often used in the design of databases. The graph-theoretic data model can be used as a conceptual model outside of a graph database. A conceptual schema is a description of the concepts and relationships in a domain. Conceptual schemas can be shared between biologists and computer scientists as communication tools, as a baseline definition of biological concepts, or as a schema definition language for development of other than graph databases. Graph-based conceptual schemas are an effective communication medium not only between biologists and computer scientists, but also between groups developing different genome databases. It is not difficult to represent a relational or object-oriented schema as a graph, and the linked tree data structure of ACeDB schemas forms a directed, acyclic graph.

Conceptual models have traditionally been used for database design, but they can also be used to define a common data exchange language between databases. Many graph-based conceptual models have their genesis in natural language processing research. These formalisms which were originally designed to model human discourse and which have proven themselves capable for describing

database schemas are also useful for exchanging data between databases. The data exchange process is to create a common conceptual schema using a graph-based conceptual data model, develop a view of the schema for each database, decompose the schema into binary relations and develop a translator between each relation or object in the database and a set of binary relations. Creating a common conceptual schema may be a difficult process in general, but it is easier in the genome domain because genome data has a graph-like structure and a common conceptual schema has already been developed for several major molecular biology databases. It is fairly straight forward to develop views of the common schema for each database and to decompose the schema into binary relations. Developing a translator between the relations or objects being used and binary relations is not difficult but can be time consuming, however, WEB allows the translation programs to be written as simple logic programs.

7. Conclusion

We have investigated using graphs as the basis for genome map databases. Genome databases are needed, and graph databases are a good way of storing and querying genome map data. Three advantages of using graph databases are:

1. Graphs are a good way of representing genomic data. Graphs closely model the interconnected nature of biological data and help integrate genome data, balancing the relative importance of genome objects and relationships.
2. Graphs have a firm foundation in mathematics and computer science. Graph theory is supported by a tremendous amount of formal analysis and study.
3. A graph-theoretic data model supports the operators needed for genome databases, including algorithms for answering complex queries.

Our investigation of graph databases suggest that graphs are a good representation for genome maps and that graph databases are sufficient to help meet the current and near future needs for mapping databases.

8. Acknowledgments

The authors thank Bob Cottingham, Dan Davison, Wayne Parrott, Joanna Power, and Randy Smith for frequent discussions of the ideas in this paper. Thanks also to Jeff Chamberlain at the University of Michigan Human Genome Center whose laboratory was modeled in the example presented in section 5. 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. A portion of this work was performed when M.G. was at the University of Michigan and supported by NSF grant ISI-9120851.

References

1. Allen, James. Maintaining knowledge about temporal relations. *ACM Transactions on Database Systems*, 15(3), 1983.
2. Brodie, Michael, John Mylopoulos, and Joachim Schmidt, editors. *On conceptual modelling: Perspectives from artificial intelligence, databases, and programming languages* Springer-Verlag, New York, 1984.
3. Cattell, R. *Object Data Management.* Addison-Wesley, 1991.
4. Cinkosky, M.J., Fickett, J.W., Barber, W.M., Bridgers, M.A., and Troup, C.D. SIGMA: A system for integrated genome map assembly. *Los Alamos Science* 20: 267-269, 1992.
5. Chen, P P. "The entity-relationship model: toward a unified view of data," *ACM Transactions on Database Systems* 1:1, pp. 9-36, 1976.
6. Codd, E.F. A relational model fo data for large shared data banks. *Communications of the ACM*, 13(6): 377-387, June 1970.
7. Consens, Mariano and Alberto Mendelzon. *The G+/ GraphLog visual query system*. In Hector Garcia-Molina and H.V. Jagadish, editors, Proc of the 1990 ACM SIGMOD International Conference on Management of Data, page 388, May 1990.
8. Cui, Zhan. Using interval logic for order assembly *Proceedings of the Second Interntional Conference on Intelligent Systems for Molecular Biology (ISMB-94)*, AAAI/MIT Press, 1994.
9. Date, C.J. *An Introduction to Database Systems Volume 1I.* Addison-Wesley, 1983.
10. Date, C.J. *An Introduction to Database Systems Volume 1, 5th Ed*. Addison-Wesley, 1991.
11. Freska, C. Temporal reasoning based on semi-intervals. *Artificial Intelligence.* 54:199-227, 1992.
12. Graves, M. *Theories and tools for designing application-specific knowledge base data models*. Ph.D. Thesis, University of Michigan. University Micorfilms, Inc. April, 1993.
13. Graves, M. Integrating order and distance relationships from heterogeneous maps. *In* L. Hunter, D. Searls and J. Shavlik, eds., *Proceedings of the First Interntional Conference on Intelligent Systems for Molecular Biology (ISMB-93)*, AAAI/ MIT Press, Menlo Park, CA, July 1993.
14. Graves, M., Bergeman, E.R. and Lawrence, C.B. Querying a Genome Database Using Graphs. In H. Lim, ed., *Proceedings of The Second International Conference on Bioinformatics and Genome Research.* World Scientific Publishing. June, 1994.
15. Guidi, J. N. and Roderick, T.H. Inference of order in genetic systems. *In* L. Hunter, D. Searls and J. Shavlik, eds., *Proceedings of the First Interntional Conference on Intelligent Sys-*

tems for Molecular Biology (ISMB-93), AAAI/MIT Press, Menlo Park, CA, July 1993.

16. Gyssens, Marc, Jan Paradaens, and Dirk Van Gucht. *A graph-oriented object databasemodel.* In Proceedings of ACM Conference on Principles of Database Systems, 1990.
17. Gyssens, Marc, Jan Paradaens, and Dirk Van Gucht. *A graph-oriented object model for database end-user interfaces.* In Proceedings of 1990 ACM SIGMOD Conference on Management of Data, 1990.
18. Hofestaedt, R. Modeling and Visualization of Metabolic Bottlenecks. In H. Lim, ed., *Proceedings of The Second International Conference on Bioinformatics and Genome Research.* World Scientific Publishing. June, 1994.
19. Honda, S., Parrot, N.W., Smith, R., Lawrence, C. An object model for genome information at all levels of resolution. *Proceedings of the 26th Annual Hawaii International Conference on System Sciences.* volume 1, 564 - 573, 1993.
20. Hull, R. A survey of theoretical research on typed complex database objects. In Jan Paredaens, editor, *Databases*, chapter 5, pages 193-256. Academic Press, 1987.
21. Ioannidis, Y. Special issue on advanced user interfaces for database systems. ACM SIGMOD Record, 21(1), March 1992.
22. Karp, P and S Paley. Automated drawing of metabolic pathways. In H. Lim, ed., *Proceedings of The Second International Conference on Bioinformatics and Genome Research.* World Scientific Publishing. June, 1994.
23. Kasper, R.T. and Rounds, W.C. A logical semantics for feature structures. In *Proceedings of the 24th Annual Conference of the Association for Computational Linguistics*, pages 235-242, 1986.
24. Kifer, M. and Lausen, G. F-logic: a higher-order language for reasoning about objects, inheritance, and scheme. In *Proc. of the ACM SIGMOD Conference*, pages 134-146, 1989.
25. Lee, A., Rundensteiner, E., Thomas, S., and Lafortune, S. An information model for genome map representation and assembly. Tech Report SDE-TR-163-93, University of Michigan, Dept of EECS, 1993.
26. Letovsky, S., Berlyn, M. CPROP: A rule-based program for constructing genetic maps. *Genomics* 12:435-446, 1992.
27. Levene, Marc and Alexandra Poulovassilis. *The hypernode model and its associated query language.* In Proc Fifth Jerusalem Conference on Information Technology,pages 520--530, 1990.
28. Mirkin, B.G. and Rodin, S.N. *Graphs and Genes*, volume 11 of *Biomathematics*. Spreinger-Verlag, 1984.
29. Nebel, Bernhard and Gert Smolka. 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
30. Ochs, R. A relational database model for metabolic information. In H. Lim, ed., *Proceedings of The Second International Conference on Bioinformatics and Genome Research.* World Scientific Publishing. June, 1994.
31. Ott, Jurg. *Analysis of human genetic linkage*. Johns Hopkins University Press, Baltimore, revised edition, 1991.
32. Thieffry, D. and R. Thomas. Logical Analysis of Genetic Regulatory Networks. In H. Lim, ed., *Proceedings of The Second International Conference on Bioinformatics and Genome Research.* World Scientific Publishing. June, 1994.
33. Thulasiraman, K., *Graphs: theory and algorithms*. Wiley, 1992.
34. Tsichritzis, Dionysios C. and Anthony Klug (eds.). The ANSI/X3/SPARC DBMS Framework: Report of the Study Group on Data Base Management Systems. I*nformation Systems* 3 (1978).
35. Shieber, S. *An Introduction To Unification-Based Approaches to Grammar.* CSLI, Stanford, CA, 1986.
36. Ullman, J. *Principles of database systems*. Computer Science Press, Rockville, MD, 2nd edition, 1982.
37. Ullman, J. *Principles of database and knowledge-base systems*, volume 1. Computer Science Press, Rockville, MD, 1988.
38. Zdonik, S.B. and D Maier, editors. *Readings in object-oriented database systems*. Morgan Kaufmann, San Mateo, CA, 1990.

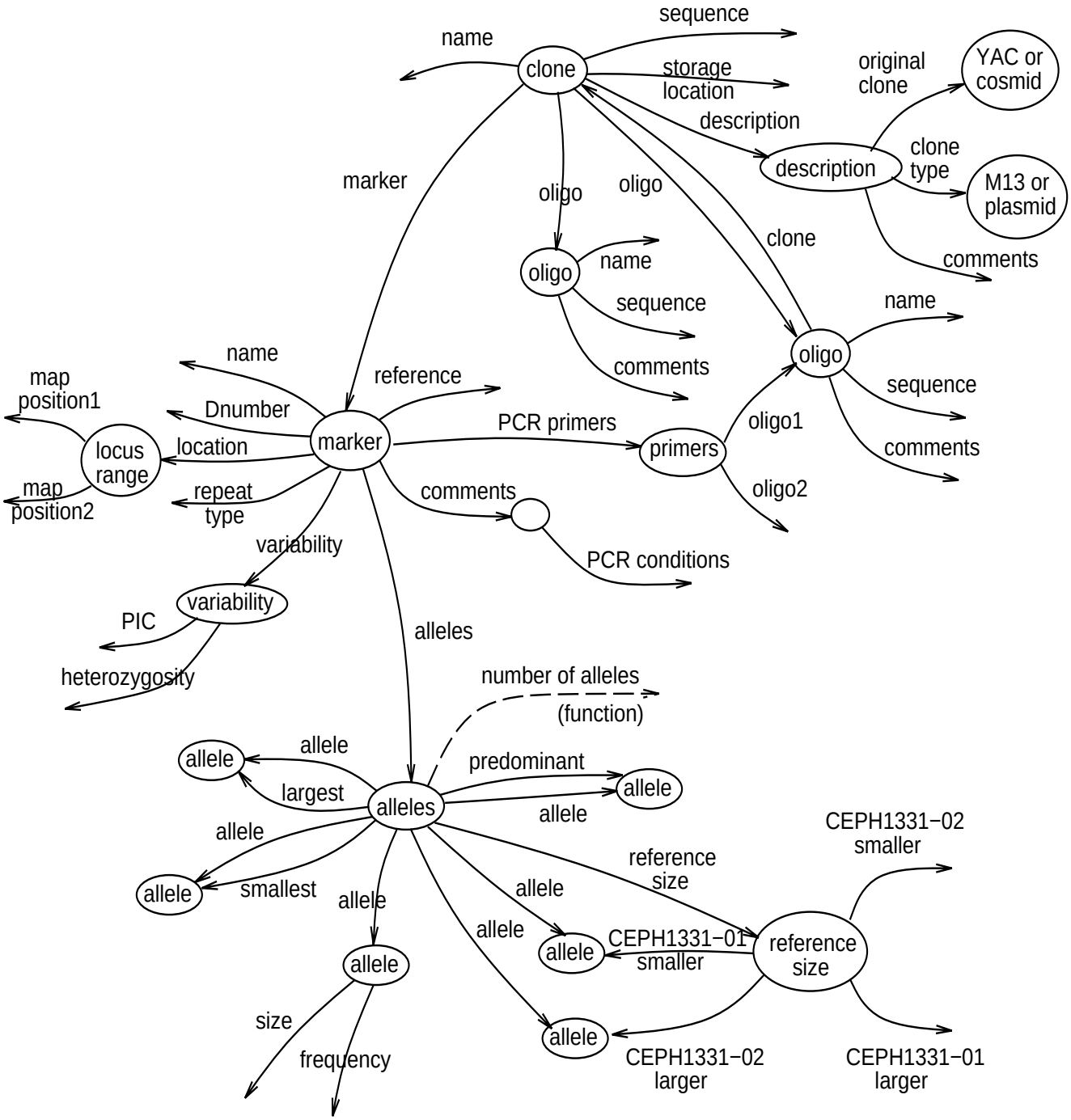


Figure 1: Relationships of a marker and a clone which contains it.