Querying a Genome Database Using Graphs

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

Departments of Cell Biology & Human and Molecular Genetics,
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)

One problem facing biology today is that information is not integrated, not connected. Graphs are a way of connecting together objects in the world: to integrate biological data. Graphs are defined by computer scientists and mathematicians as a collection of nodes and edges --- real world objects and connections between them. For graphs to be useful, they must be tailored to the needs of geneticists. We have tailored the definition of graphs to support the requirements of genome mapping databases at the Baylor College of Medicine Human Genome Center and present here how graphs can be used to query a genome database.

We discuss the requirements for a genome database, describe a graph database, show how genome data can be stored as graphs, and present query algorithms and interfaces which are useful for querying genome data.

1. Motivation

The Human Genome Project is one of the grand challenges of the scientific community. Already millions of dollars and hours have been spent in laboratories as scientists work toward mapping the human genome.

Current genome research is oriented toward developing maps of genome-derived reagents at different levels of resolution. The genome reagents include chromosomes (normal and abnormal), clones of subregions of genomic DNA maintained in a variety of cloning vectors, cDNA clones derived from different tissues, and markers for polymorphic sites in genomes from populations of individuals. Common genome maps include: cytogenetic maps which localize a reagent to a subregion of a chromosome; genetic maps which order polymorphic genetic markers and can be used to find the approximate chromosomal location of genetic material responsible for an inherited trait; physical maps which provide a physical ordering of reagents relative to a chromosome; and

DNA sequence which is a type of high-resolution physical map. Such maps are being developed for a number of diverse species including human, mouse, *Drosophila*, *C. elegans*, budding yeast, fusion yeast, *Arabidopsis*, many prokaryotic genomes, and several agriculturally important plant and animal species.

As these genome projects progress, more attention will be turned to identifying: the location of all genes and the genome components that regulate their expression; the biochemical function of gene products including their role in metabolic and regulatory pathways, and interactions with other gene products; the pattern of individual gene expression during development; the subcellular location of gene products; and the evolutionary history of genes. As this data corpus grows, biology-related research will be facilitated by computational tools which integrate genome and biological data in such a way that it can be explored in a flexible and meaningful way by researchers.

A scientist would not expect to work with outdated laboratory equipment or continue to use outdated techniques if better methods and machines were available, yet the data management support which is an integral part of the project is still based on 1970s computer science. This does not mean that the existing systems do not support current needs, but rather that there are new and better data management systems which should be considered alongside the systems which are currently accepted by the genome community.

In the Human Genome Center at Baylor College of Medicine, we are motivated by the need to capture and integrate data at virtually all possible levels of genomic resolution generated within the Center, and to integrate this data with external databases that contain related information important to our research. Over the past year, we have extensively analyzed the requirements for an information management system which will support our research needs. One result of this analysis is the realization that storing genome research data in a data structure based on graph theory will facilitate meeting many of our design goals for integrating data and developing effective end-user applications to query and browse the data.

Developing a database system which can evolve, be integrated with other systems, be user accessible and maintain data integrity is the goal of the group developing this project. Having spent time learning about the real needs of genome researchers and the complexity of the genome data, we believe that graph databases will be the best tool for data management in the future of the genome project.

We describe graph databases, show how genomic information can be represented as graphs, and describe query algorithms which allow complex queries to be asked of a graph database.

2. Graph Database

A database consists of three architectural levels: external level, conceptual level, and the internal level (Tsichritzis *et al*., 1978; Date 1990). 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. 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. The database implementor builds a system which supports the requirements for the internal level and efficiently stores the data in persistent (non-volatile) storage device.

We describe a graph database by defining it at the internal, conceptual, and external levels.

2.1 Binary Relationships

A *binary relationship* links two objects through a relation (Figure 1). The internal level of the

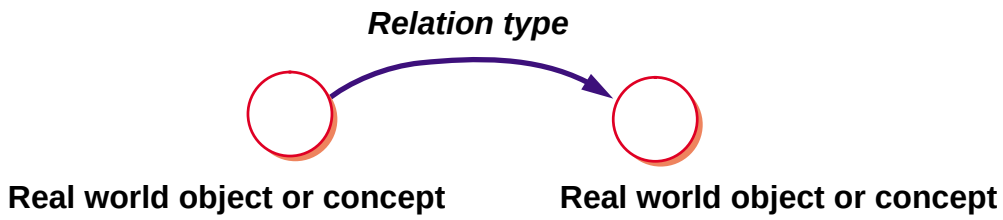

Figure 1. A binary relationship consists of a relation having a specific type that directionally links two objects.

graph database is defined by specifying how the binary relationships that make up graphs are stored and retrieved. We have developed 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 asked more efficiently. Each binary relationship query consists of an index table lookup.

2.2 Graph-Theoretic Data Model

A graph-theoretic data model is a data model which 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 (Levene, 1990) and allowing edges to relate to other edges (Graves, 1993). Related data models include GOOD (Gyssens, 1990), G+/GraphLog (Consens, 1990), and Hyperlog (Levene, 1990). Our graph-theoretic data model extends the definition of a graph to include other features, such as labeling edges with types, packaging other graphs as an encapsulated vertex, and graph querying algorithms. These added features make our graph-theoretic data model more useful for modeling genomic data than other graph-theoretic data models.

We have also developed a query language called WEB which implements some of our operators and can be used to query a graph database (Graves, 1993). 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. WEB programs define graph templates, which are described in the next section.

2.3 Graph Templates

The user views in the external level are means by which end-users and application programmers interact with the database. The views allow access of genomic objects and relationships stored in the database. Our approach to creating user views is based on *graph templates*, programs written in WEB that specify parameterized graphs. Graph templates are created from the database schema and user views and used as interactive views on a database. They define what the data in the database looks like to the end user. Graphical depictions of WEB programs defining graph templates are given in Figure 2. Each of the arcs in the template would be labeled with a variable in a WEB

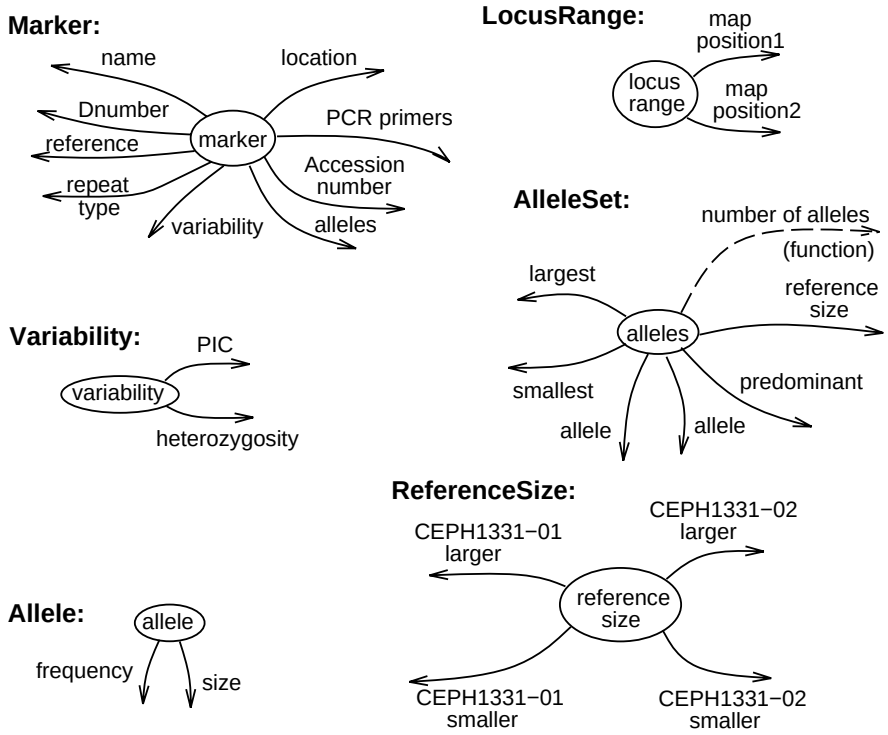

Figure 2: Some graph templates for genome markers

program.

The WEB program for the view of allele in Figure 2 is:

allele(?frequency, ?size) ≡ <frequency, ?allele, ?frequency>, <size, ?allele, ?size>

where “?name” is a variable, “?allele” is the variable bound to “allele” node, and each binary relationship in the program “<relation, source, destination>” is an arc in the graph.

There are two kinds of graph templates: (1) those for creating, manipulating and querying objects and relationships in the database and (2) those for querying only. The mechanisms used for representing both types of graph templates are identical. Because WEB is a declarative programming language (like Prolog), a program in WEB can be used for both creating and querying. The schema developer is responsible for creating templates and specifying them to be used (1) for creating, manipulating, and querying objects or (2) for query-only. The query-only templates describe an incomplete view of the database and should not be used for entering or manipulating data.

End users use query-only graph templates to query the database via a form-based or graphical user interface. Application program use both kinds of graph templates as an application program interface to interact with the database.

To facilitate the development of application programs, application views encapsulate all graph templates they use as *graph template objects*. Graph template objects are WEB programs with access methods and inheritance which define an interface between the database and the application program.

The database developer creates graph template objects as support to the application design process. A graph template object specifies the definition, manipulation, and retrieval of a specific graph, and appears to the application developer as an external system interface which can be called from application programs. In our implementation based on object-oriented technology, the application program interface appears as a collection of objects. All interaction between the application and database takes place through graph template objects. This restriction provides for data integrity and security.

Our experience has been that using graph templates improves the reliability of the design and leads to the development of domain objects which are more stable as the application evolves. It appears that this occurs because the natural modeling of genomic concepts and relationships as graph makes possible inconsistencies apparent more quickly than in traditional design methodologies.

3. Storing Data

Data is stored in the database as binary relationships. To define the necessary binary relations, we have developed a methodology for using graphs to describe domains:

1. Create graph descriptions of the domain.

2. Choose useful subgraphs which correspond to concepts and relationships in the domain.

3. Define graph templates which specify the subgraphs.

4. Define graph templates as objects which are the domain objects in a database application.

Domain design is done using graphs to capture concepts and relationships. Each concept within a domain should be a node in the graph and all relationships between concepts should be drawn as arcs. Any relationship within the domain should be captured in the model, without concern for its relative importance within the database. Since graphs do not have complex modeling syntax, the designer is free to concentrate on capturing all information without making analysis and design decisions. In the process of domain analysis and defining graph templates, we have found that creating use-cases and scenarios help to capture user views more completely and accurately, and that graphical descriptions are useful tools in communicating with domain experts. This leads to a more complete understanding of the domain.

After the developer has been satisfied that the domain is reasonably complete, she can then start to determine the relative importance of nodes in the graph. Subgraphs which describe important con-

cepts within the domain will be apparent and should be distinguished. These subgraphs will be used in the process of defining graph templates.

Graph templates are defined to specify subgraphs corresponding to useful concepts. Each graph template specifies a single view onto a complex genome concept. Since each complex concept can have many relationships which describe diverse views, it is necessary to define a separate templates for each view. While relationships may overlap between views, each view should describe a unique concept within the domain.

Application domain objects are subgraphs which were defined during application analysis and design. They are encapsulated into objects for application development. Applications may share common subgraphs or use uniquely defined subgraphs, depending upon the needs of the application domain as defined by the developer.

4. Querying

In addition to storing data using graphs, we can also use graphs for querying. To do this, a user defines a query using one of several user interfaces. The user interfaces specify graph templates which incorporate the desired query. Each graph template can define several queries depending upon which parameters to the graph template are bound. In a specific query, some of the arguments to graph templates may be specified and some are left unbound. The partially specified graph templates are given to a graph query algorithm which retrieves matching graphs from the database. The matching graphs are passed to a report generator which formats the graphs into a report for the user.

Traditionally, databases have been developed which have a pre-defined structure. Domain concepts and relationships are forced into that structure, and the user is 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 the concepts and relationships in the domain, and allow the user to ask queries where the structure of the query follows closely the organization of information in the domain.

4.1 Query Algorithm

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 retrieved from storage are combined into graphs as specified by the query graph. These resulting graphs are the output 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 supports 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 supports the specification of a theoretically complete set of queries and is based on the structure of the graph-theoretic data model. An advantage of the two engine approach is that the first query engine can be updated more easily as users ask more complex queries and as the types of genome maps evolve. Initially, we will develop 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.

Both query algorithms are based upon the four step process above. They place different restrictions on the query graph (a rooted directed acyclic graph versus a directed graph) which simplifies the algorithm used to build the final result from the collection of binary relationship query results.

The rooted, directed, acyclic graph query engine decomposes rooted DAGs into a tree and equality constraints, decomposes the tree into a set of paths, and a path into a sequence of binary relationships. The full graph engine decomposes a graph into a collection of binary relationships. Thus, there are five querying algorithms: binary relationship, full graph, path, tree, and rooted, directed acyclic graph (DAG).

A binary relationship query is the interface between the conceptual and internal levels of the database, and all graph query algorithms access the stored data through binary relationships. A binary relationship consists of a relation, a source, and a destination. Any one of the three can be either specified or unspecified in a query. Thus, there are 2^3 different kinds of binary relationship queries.

We describe the full graph query algorithm and the tree and rooted DAG query algorithm in the next two sections.

4.1.1 Graph Querying

A graph is a collection of binary relations. A graph can have cycles or be disconnected. A graph query is defined by a graph template which is a graph with some of the nodes and edges replaces with variables.

The graph 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 works by comparing 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 graph query algorithm decomposes 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 order in which the binary relationships are queried against the database does not matter, so an arbitrary order is chosen.

The algorithm takes the first binary relationship of the ordered set and queries it against the database. The binary relationship can have 0, 1, 2, or 3 variables. The values found in the database for the variables are stored in a temporary table. The process is repeated for the second binary relationship. The two temporary tables can share 0, 1, 2, or 3 variables. A join is taken between the two temporary tables to create a new temporary table. The third binary relationship is queried and its result is joined to the third temporary table creating another temporary table. This last step is repeated for each binary relationship in the query. The final temporary table is the result of the query.

The implementation of the algorithm reuses temporary tables and suspends cross products until needed to improve the space efficiency of the algorithm. The query algorithm is completely implemented in Common Lisp and is described in more detail in Graves (1993).

4.1.2 Rooted DAG Querying

Querying a arbitrary graph is not always necessary. By restricting the structure of the graph, more efficient query algorithms can be developed. Two useful restrictions are trees and rooted, directed acyclic graphs (DAGs). A tree is a collection of paths with a common source. A rooted DAG (sometimes called a rational tree) is a tree where some of the leaves are constrained to be shared.

The basis of the rooted DAG and tree query algorithms is the path querying algorithm. A path is a sequence of binary relations where the destination of one binary relation is the same as the source of the next one. The end nodes of the path are defined by the source of the initial binary relationship and the destination of the final binary relationship. For example, the sequence of plasmid in a library would be a path from library to plasmid to the sequence.

The tree query algorithm is not particularly interesting alone. Its primary purpose is to function as a foundation for the rooted DAG query algorithm. A tree query is performed as a collection of path queries where the root of the tree is the shared initial source of the paths. A rooted DAG query is decomposed as a tree with a collection of equality constraints on the leaves and performed as a tree query. The resulting trees whose leaves do not meet the constraints are omitted from the rooted DAG result.

The rooted DAG query engine is an algorithm based on path following extended to allow for equality constraints on the paths (Kasper and Rounds, 1986). The algorithm works by following 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 allow queries to specify that two paths must point to the same object. This algorithm is related to algorithms currently being used for natural language processing systems (Shieber, 1986) and knowledge representation and reasoning systems (Nebel, 1990) but modified to query against a database.

A query is formed as a collection of paths in the graph. This forms a new, query-only graph template which is rooted at a specified object. For example, the query:

Find probes for genetic markers with heterozygosity > 0.7.

can be queried against a database which is partially described by the view schema in Figure 3. A

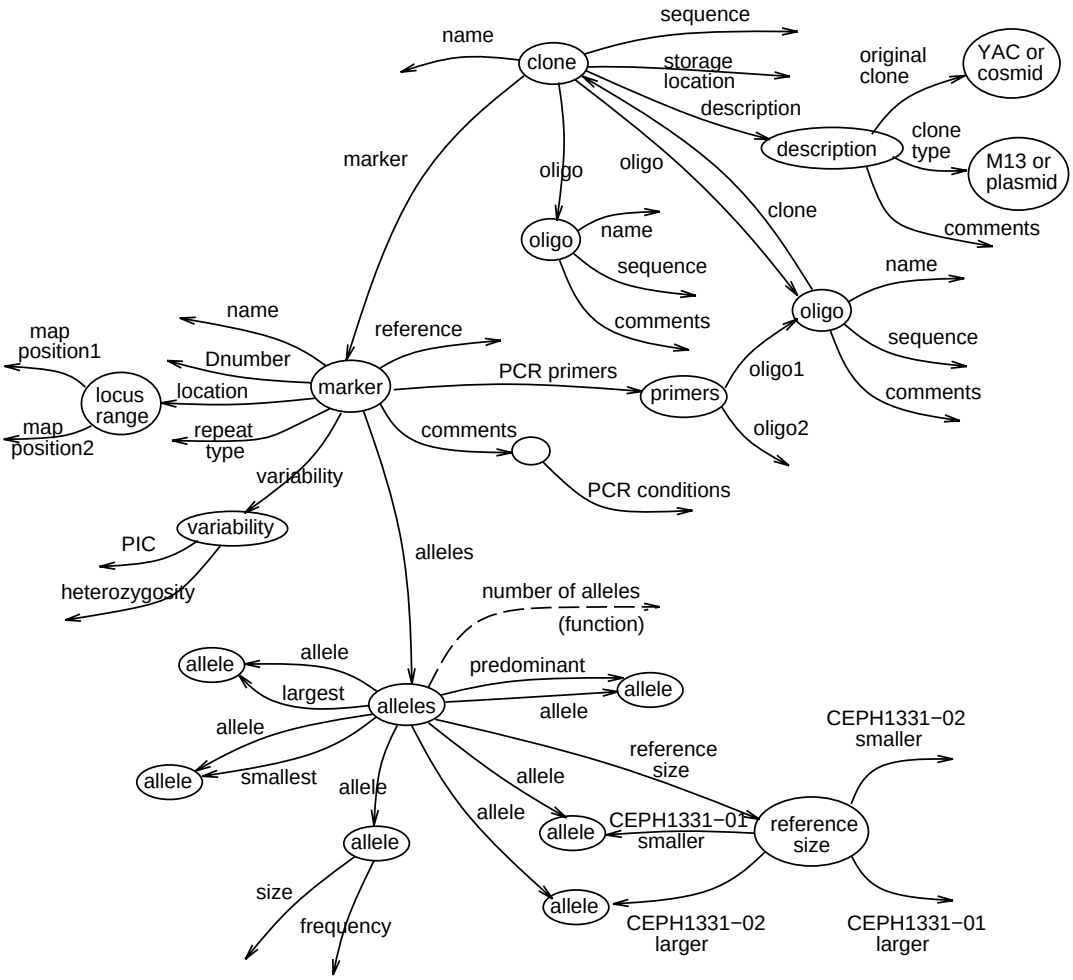

Figure 3: View schema of a PCR marker

rooted DAG is specified as a subset of the schema graph which answers the query. An advantage of this algorithm is that queries can be readily specified using text-oriented user interfaces as well as graphical ones. This allows for querying using text-based tools such as Mosaic. A graph template for the query can be specified as a query object:

Table 1: Query Object for Marker Graph Template

Role	Path
name	name
Dnumber	Dnumber
location1	location.mapposition1
restrictions	heterozygosity > 0.7
constraints	none

Table 1: Query Object for Marker Graph Template

Role	Path
location2	location.mapposition2
heterozygosity	variability.heterozygosity
primer1	PCRprimers.oligo1.name
primer2	PCRprimers.oligo2.name
sequence1	PCRprimers.oligo1.sequence
sequence2	PCRprimers.oligo2.sequence
restrictions	heterozygosity > 0.7
constraints	none

The roles are used to track the paths and display the results in reports. Restrictions can be added to any role to restrict the values which can occur at the end of the path. Constraints can be placed on two paths that they meet at the same item. These equality constraints are different than placing a restriction on two paths that they have the same value and are useful to "connect" different parts of the query graph. The query object retrieves a query based on the subgraph in Figure 4.

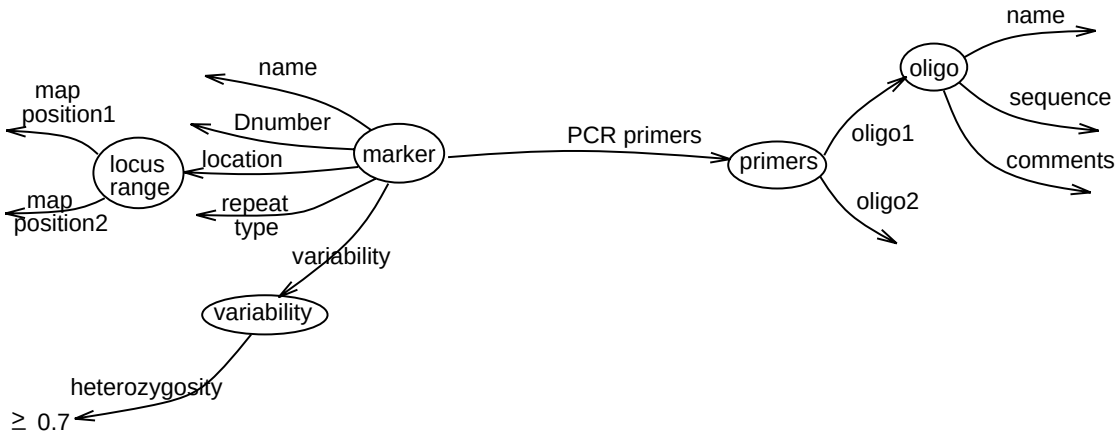

Figure 4: Rooted DAG query on a PCR marker

4.2 User Interface

One of the advantages of a graph-based approach to databases is the ease at which graphical user interfaces can be designed. However, the most obvious presentation is not always the best one, and we have found that free-form graphical input may not always be appropriate. We have developed a prototype form-based query tool for binary relationships and are currently developing a graphical user interface for complex genome queries. Query tools usually present the query result as a report, and we are extending this by allowing the report to be a hypertext document.

4.2.1 Binary Relationship Query Tool

Binary relationships can be queried using a form-based user interface. The binary relationship query tool uses a form-based user interface to query binary relationships and is shown in Figure 5.

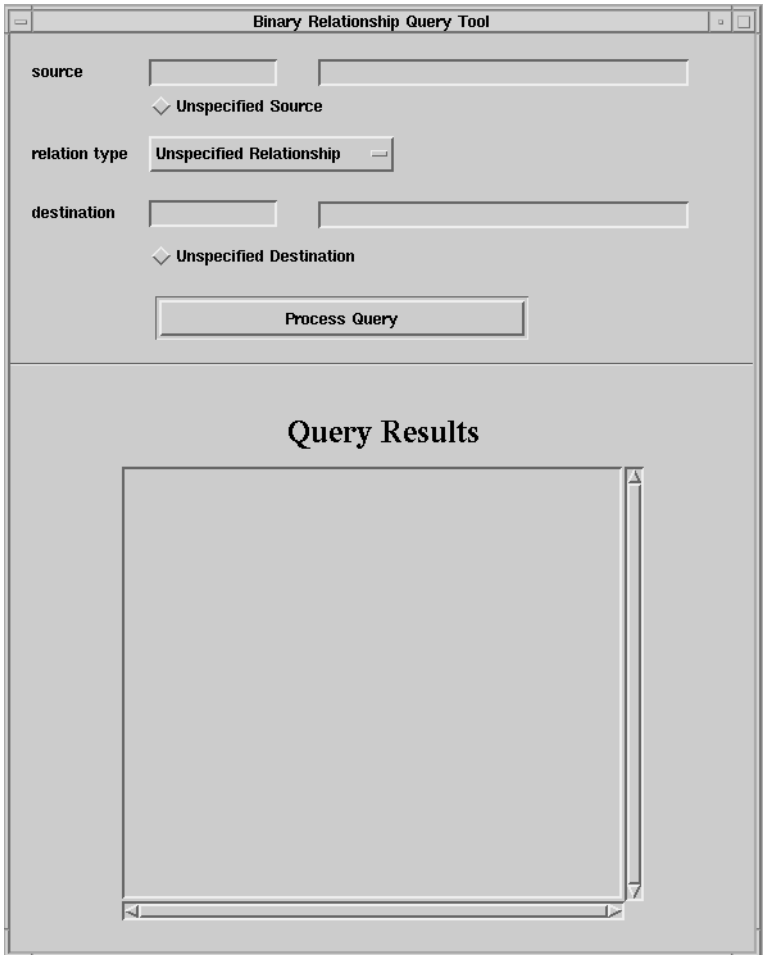

Figure 5: A binary relationship query tool.

A prototype of this tool has been implemented in Smalltalk, and we plan on implementing a Mosaic version.

In our prototype interface, the user can either select a source, destination and relationship to query or leave any of them unspecified. If a source or destination is needed, the first blank is for the type of object that will be queried and the second blank is for the specific data item of interest.

To prevent invalid queries from being created, the query tool would only allow the user to select relationships that are appropriate for the source and destination objects. In the same way, object types and data items will be checked as the user creates the query.

4.2.2 Graph Query Tool

Graphs queries are defined as a collection of binary relationship queries. Instead of using WEB directly to define the queries, a graphical user interface is used. An end user begins with one or more

graph templates which define user views. These schemas can be combined and edited to create a new graph template object. These query-only graph template objects are passed to a query operator in the conceptual level of the database. The query is evaluated by the query operator which calculates the query result and returns the result to the external level. This interface will use the full query engine to retrieve graphs from the database and generates reports into ASCII text files, Reportoire documents (a commercial product), or hypertext markup language (HTML) documents. The initial design of this tool has been completed, and we are currently developing a prototype implementation.

A graphical user interface based on using graph templates which represent domain objects and relationships is shown in Figure 6. From the list of Genome Objects, the user selects one or more to

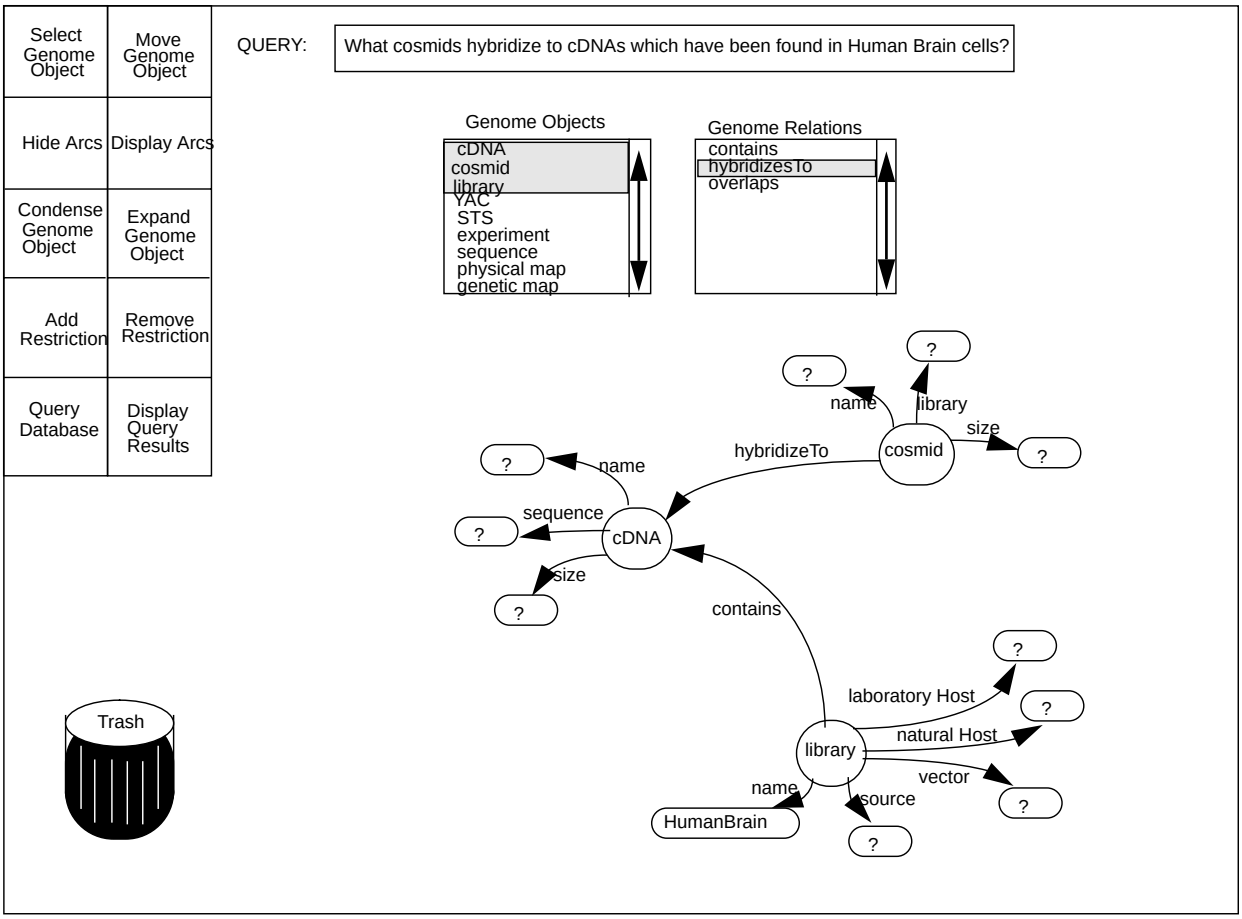

Figure 6: Graph query tool

be used in the query. As each object is selected, the graph template for the object is drawn on the screen. Each template can be selected with the mouse and moved around the screen. When the user selects an object to use within a relation, the Genome Relations list changes to display all relations in which the selected genome object can participate. After the user selects a relation from the list, a labeled arc is drawn from the selected object but not connected to any other object. Using the mouse, the user connects the loose end of the arc to the appropriate object. If the connection is not valid within the database, an error message is displayed and the arc is not connected. After all of the relations are added to the objects and restrictions are added, the user queries the

database by selecting the ‘Query Database’ button. ‘Display Query Results’ allows the user to select the way that reports will be displayed.

The user can click on ‘hide arcs’ if he does not want the selected arcs to be used in the query or shown in a report. Choosing ‘display arcs’ adds any hidden arcs to the query. If the display is crowded, the user can select a genome object and choose ‘condense genome object’ to display the genome object as a single node. Restrictions on a query can be added or removed by selecting an object and using an object specific editor to enter the appropriate information.

4.3 Report Generation

The query interface generates reports in either text or hypertext. Text reports can be written to a file, printed, or viewed interactively. Hypertext reports can be viewed using a hypertext viewer such as Mosaic. Using hypertext for reports adds flexibility to the database, because a database browsing capability is added to querying. The end user can begin with a hypertext report and venture off to explore the rest of the database.

Query tools generate reports which present information that would not be readily apparent from browsing the database. Although browsing allows for retrieval of stored facts, a flexible querying capability is important for discovering information which would not be stored as single facts. An example is a list of all markers on a chromosome with heterozygosity above a certain level. The query tools can generate hypertext reports like those shown in Figure 7. In addition, query reports function as one starting point for browsing the database. An end user can ask a query, which collects together information of interest, and then use the resulting hypertext report as a place to begin exploring the database using the browsing facility. By integrating the querying and browsing functionality, simpler queries can be asked whose result can be explored in detail rather than forcing the user to ask much more complex queries which include only the data desired.

A hypertext browser for the entire database can be defined by giving each object in the database a hypertext page and using relations between objects as hypertext links. It would probably be difficult for a user unfamiliar with the data to navigate this document, so entry points into this hypertext web are defined by an *ad hoc* query tool. The user is then free to explore only parts of the report which are of specific interest.

5. Conclusion

We have investigated using graphs as the basis for genome databases. Three advantages of using graph databases are:

1. Graphs closely model the interconnected nature of biological data.
2. Graphs help integrate genome data, balancing the relative importance of genome objects and relationships.
3. Graphs support algorithms and interfaces for complex queries.

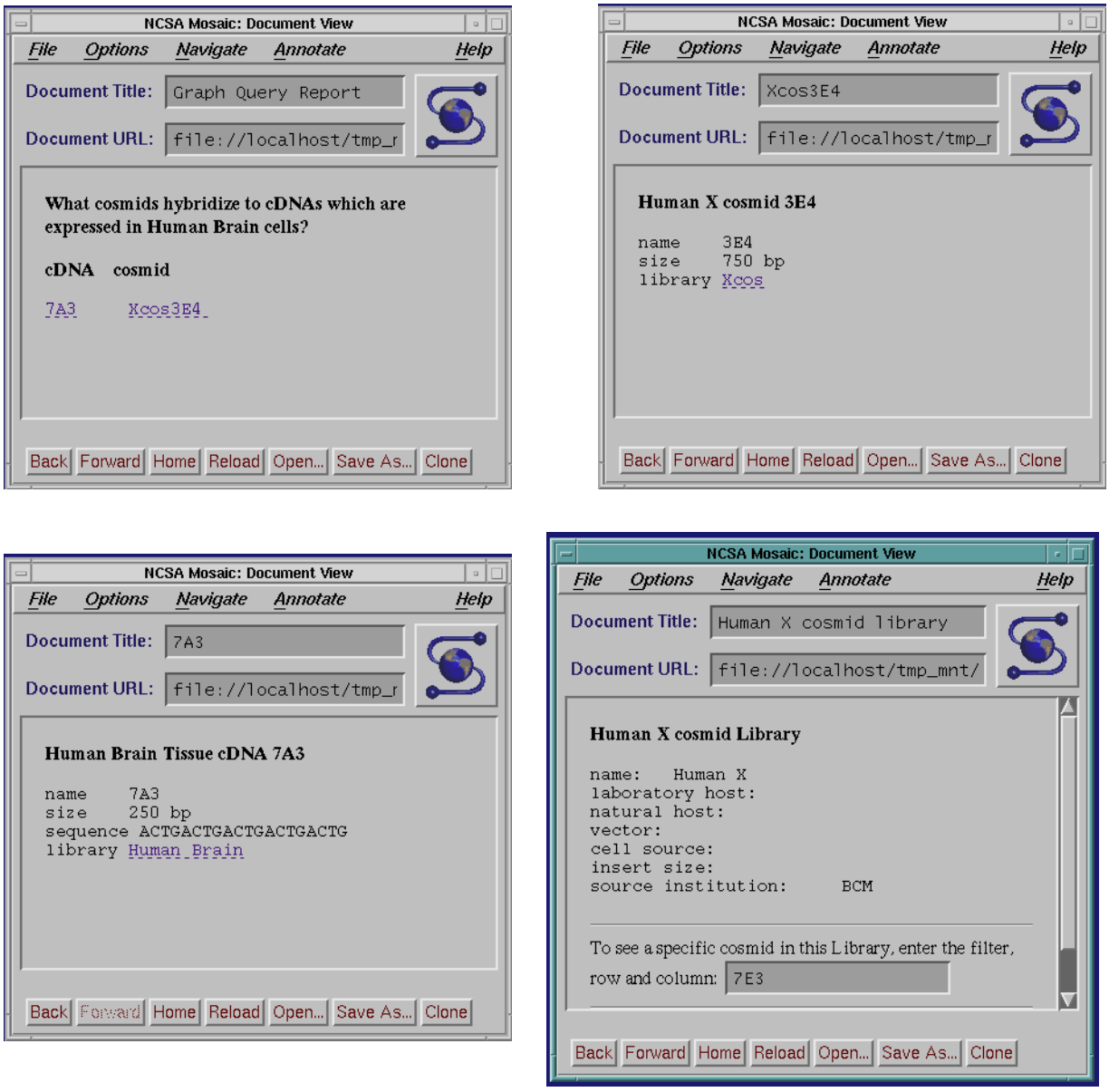

Figure 7: Hypertext reports

Preliminary results from the development of a prototype graph database suggest that graphs are a good representation for genome data, and we believe that further research will support the thesis that graph databases are sufficient for current and near future genome databases.

6. Acknowledgments

The authors thank Bob Cottingham, Dan Davison, Wayne Parrott and Randy Smith for frequent discussions of the ideas in this paper. This work was supported by a grant to C.B.L. from the Department of Energy, the Baylor Human Genome Center funded by the NIH National Center for

Human Genome Research, a fellowship to M.G. from the National Library of Medicine, and the W.M. Keck Center for Computational Biology.

References

1. Consens, Mariano and Alberto Mendelzon.*The G+/GraphLog visual query system*. In Hector Garcia-Molina and H.V. Jagadish, editors, Proc of the1990 ACM SIGMOD International Conference on Management of Data, page 388, May 1990.
2. Date, C.J. *An Introduction to Database Systems Volume 1, 5th Ed*. Addison-Wesley, 1991.
3. 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)*, Menlo Park, CA, July 1993. AAAI/MIT Press.
4. Graves, M. *Theories and tools for designing application-specific knowledge base data models*. University Microfilms, Inc. Ph.D. Thesis, University of Michigan, April, 1993.
5. 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.
6. 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.
7. 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.
8. Mirkin, B.G. and Rodin, S.N. *Graphs and Genes*, volume 11 of *Biomathematics*. Springer-Verlag, 1984.
9. 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
10. Thulasiraman, K., *Graphs: theory and algorithms*. Wiley, 1992.
11. 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).
12. Shieber, S. *An Introduction To Unification-Based Approaches to Grammar.* CSLI, Stanford, CA, 1986.