VitaPad: visualization tools for the analysis of pathway data

Matthew Holford¹, Naixin Li¹, Prakash Nadkarni² and Hongyu Zhao¹,³,*

¹Center for Statistical Genomics and Proteomics, ²Center for Medical Informatics and ³Division of Biostatistics, Yale University, New Haven, CT 06520, USA

ABSTRACT
Motivation: Packages that support the creation of pathway diagrams are limited by their inability to be readily extended to new classes of pathway-related data.

Results: VitaPad is a cross-platform application that enables users to create and modify biological pathway diagrams and incorporate microarray data with them. It improves on existing software in the following areas: (i) It can create diagrams dynamically through graph layout algorithms. (ii) It is open-source and uses an open XML format to store data, allowing for easy extension or integration with other tools. (iii) It features a cutting-edge user interface with intuitive controls, high-resolution graphics and fully customizable appearances.

Availability: http://bioinformatics.med.yale.edu
Contacts: matthew.holford@yale.edu; hongyu.zhao@yale.edu

INTRODUCTION: EXISTING TOOLS AND THEIR DATA MODELS

Biological pathways are networks of relationships between biological entities (http://www.biopax.org). A pathway can represent several kinds of biological phenomena, such as signal transduction or metabolic transformation of particular molecules: the scope of a pathway may be within a single cell or sub-cellular component (e.g., the citric acid cycle, which occurs in the mitochondria) or across several organ systems (e.g., drug biotransformation and elimination). While pathway discovery itself is a scientific process that is at least a century old, the systematic cataloguing of pathways and their storage in database form has recently gained increasing importance with the realization that knowledge of pathways greatly facilitates interpretation of high-dimensional experimental data, such as gene expression data generated by microarray experiments.

As the variety of experimental data that can be brought to bear on pathway analysis increases, and as the definition of ‘pathways’ itself broadens in scope to include multiple organ systems as opposed to a single cell, pathway analysis, management and rendering software must adapt to these circumstances. Ideally, it is desirable if the software can be designed in such a way as to be adapted by its users without having to completely rewrite the code. This is possible if the software is driven by ‘metadata’—information that informs the code about the types and structure of the kinds of data that it is dealing with.

A wide variety of systems exist for the display of pathway information, and we now review several of them from the viewpoint of purpose, diagram-rendering capability and data model. The system with the widest use is KEGG (Kanehisa and Goto, 2000). While KEGG uses static approaches for data visualization, its value lies primarily in the breadth of its content coverage. The benefits of first-generation systems such as WIT (Overbeek et al., 2000), MPW (Selkov et al., 1998) and EcoCyc (Karp et al., 1999), which perform limited dynamic visualization (rendering the diagram from the database’s contents) also accrue mainly through their content. General purpose systems for pathway rendering include BioJake (Salamonsen et al., 1999), PathFinder (Goessmann et al., 2002), Pathways Database System (Krishnamurthy et al., 2003) and PaVESy (Ludemann et al., 2004). The BioJake website is not operational currently: the paper indicates that the system was, at best, an early prototype. PathDB’s data model (http://www.ncgr.org/pathdb/doc/schema.pdf), while rich, is not user-extensible: this limitation also applies to the Pathways Database System. PathFinder is a versatile program that seeks to improve the quality of annotated data by identifying graph segments (‘chunks’), scores for which are lowered if particular enzymes that are expected to be in the pathway are missing from a particular set of annotated data. While its data model is restricted, such a restriction appears to be appropriate to its purpose. PaVESy uses a potentially extensible Entity-Attribute-Value database model: however, its schema lacks the metadata components that support such extensions in a robust, error-free fashion.

Several recent papers have noted the importance of presenting microarray data in the framework of documented biological pathways (DeRisi et al., 1998; Becker and Rojas, 2001). To this end, researchers have developed a handful of applications that can incorporate microarray data with pathway diagrams (e.g. Pan et al., 2003; Rhee et al., 2003). Of these, the most significant and widely used is GenMapp (Dahlquist et al., 2002), a Windows application allowing users to create and modify pathway diagrams and color them according to their experimental data.

Because of the extremely wide scope of biological pathway-associated phenomena, existing efforts to create data exchange formats for pathways have had to restrict themselves in various ways. The current version of the BioPAX ontology (v 0.5.2), for example, is restricted to definitions that apply within the scope of a single cell. It does not consider multiple categories of cells within an organ (such as the liver or intestinal mucosa), or pathways that extend across organ systems, such as those involved in pharmacokinetics. Such
Visualization tools for the analysis of pathway data

limitations of scope complicate both the task of devising a generic and extensible database representation of pathway-related information, as well as the task of rendering the information within a database for an arbitrary pathway as a diagram.

Another feature of existing software for composing pathway diagrams is that its ‘drawing palette’ is hard-coded with the concepts—genes, receptors, etc., that form the ‘primitives’ of some (but not all) pathway biology. While this is a convenience for diagram composition, the flexibility of the software becomes greatly limited—it is not easy for a user of the package to extend it to incorporate a new class of data that the package may not have accommodated.

Database-driven diagramming tools that are not arbitrarily limited with respect to the types of data that they can manage will therefore facilitate informatics support for pathway-related research. We describe such a tool, VitaPad.

SYSTEM FEATURE SUMMARY

(1) VitaPad uses a database design that separates the biological aspects of pathway-related knowledge from the knowledge of how individual pathway entities, or classes of entities, are to be rendered in diagrams. This design is intended to be extensible, so that new classes of data that VitaPad did not originally know about may be added to the schema.

(2) VitaPad’s pathway-rendering process deals only with abstract one- or two-dimensional shapes, so that concepts like genes, enzymes or substrate/product molecules are not hard-coded in the rendering algorithm itself. Instead, the rendering process utilizes specifications that indicate the correspondence between a particular kind of shape and a particular class of biological entity such as a gene.

(3) The loose coupling between biological knowledge and rendering knowledge allows new classes of data to be rendered on diagrams. All that is required is to create a rendering specification for each new class. At startup, database tables that contain these specifications are read and used to customize the drawing palette with icons representing each data class.

(4) VitaPad currently leverages Graphviz (Gansner and North, 1999), a sophisticated open-source graph-drawing package developed by computer scientists at AT&T Research, to automatically lay out a pathway diagram. The algorithms in Graphviz have been devised for general-purpose rendering of directed or undirected graphs on a plane surface, and are not specific to pathways. They are quite robust—for example, they will detect loops in the graph, such as would be expected in cyclical metabolic pathways. The automatically generated layout may not, however, be exactly what the user wants, and so VitaPad allows individual objects to be rearranged in the diagram: the adjusted positions of the pathway objects can be saved for later reuse.

(5) The database that supports VitaPad is IPA (Integrated Pathway Analysis), an Oracle database developed by our group and accessible at http://www.bioinformatics.med.yale.edu, which contains reaction data from over 100 pathways each for Homo sapiens, Mus musculus and Rattus norvegicus, as well as expression data generated by our research collaborators. The VitaPad code that accesses this database, however, is DBMS-independent, using standard SQL (through JDBC) rather than the Oracle-specific dialect of SQL (PL-SQL). The database can therefore be ported to another DBMS, such as MySQL, without requiring modification of the VitaPad code.

(6) In addition to letting one compose pathway diagrams through a drawing palette, VitaPad can also import pathway data via an XML-based input format. In addition, it can write to this format.

(7) VitaPad allows three levels of graphics customization. At a global level, one may define the appearance (‘style’) of a class of data: these settings will be used as the default when VitaPad starts up. However, in circumstances where multiple users share the same VitaPad installation, a user working on a particular pathway can define default styles for one or more classes that apply only to that specific pathway and will override the corresponding global settings. In addition, the style of individual objects may also be set. This is used, for example, to emphasize certain objects (e.g., key enzymes or molecules) over others of the same class in a given pathway.

(8) By making the source code for VitaPad, which is written in Java, freely available, we intend that the software should be deployable on a wide variety of platforms, so that it may be adapted by researchers to fit their specific needs, such as integration with other software or incorporation of new classes of data.

USER INTERFACE

A VitaPad graph is made up of instances of three basic elements: vertices, edges and decorations. Edges connect vertices, while decorations are affiliated with edges and can be said to modify them. An edge must connect two vertices and may optionally have any number of decorations. For metabolic pathways, vertices correspond with stable chemical compounds located along the pathway; edges, the reactions between them; and decorations, the enzymes or genes catalyzing these reactions. A second decoration is used to represent the expression value for each enzyme or gene under the specified experimental conditions (Fig. 1). Each data element contained
Fig. 2. A screenshot of VitaPad’s point-and-click graphical editor. The picture shows part of the nitrogen metabolism pathway in humans (pathway details originally obtained from the KEGG database).

within the graph has an analogous visual element which determines its appearance and whose properties are customizable.

There are two views for display and user interaction: the graph view (Fig. 2), a point-and-click visual editor, and the table view, a spreadsheet-style editing framework that lists the individual objects. Though the views provide the same editing capabilities, the graph editor offers easier control over the final appearance of the graph while the table editor gives the user the convenience of seeing all the graph data at once. The table view also allows the user to disable elements they may not wish to show on the final graph.

Automatic pathway drawing is performed by using the contents of the database to generate input to ‘neato’, a component of the GraphViz library that renders undirected graphs on an X–Y plane. All the algorithms in this library attempt to optimize the positions of vertices such that vertices that are conceptually adjacent are rendered accordingly, and such that the number of edges that cross each other in the plane is minimized. For this, they use an optimization process that, in principle, treats the edges as springs whose tension is proportional to their length in the diagram, and try to minimize the overall tension in the diagram. Graphviz itself can render the diagram in a variety of graphics formats, as well as allow customization of the graphic in various ways; e.g., to specify shapes, sizes and colors of individual objects. We use it, however, in a more restricted mode, where the algorithmic output (the X–Y positions of the vertices) is passed back to VitaPad, which then performs the actual rendering, using simple layout rules such as attempting to space decorations on an edge evenly.

VitaPad supports graphic display at various magnifications—the diagram may be considerably larger than the physical screen, and the user can pan within the graphic. The diagram-editing interface takes advantage of the knowledge of connections between elements to make editing faster and easier. Thus, for example, manually moving a vertex automatically repositions the edges that are connected to that vertex and this in turn automatically repositions the decorations associated with each edge.

Manual editing is primarily mouse-based and relies on a toolbar and a variety of editing dialogs. The editing menus allow the user to search through thousands of enzymes, genes and compounds referenced in the IPA database. New elements can be added if they are not found. Visual editing menus provide the capability of altering the visual appearance of any graphical element, e.g. color, shape, font, in order to create distinctive and meaningful diagrams. Thumbnail sketches of graph elements offer the user a preview of how their visual types will appear. Custom visual types may be saved for future use.

To display microarray (gene expression) data, a researcher can either select an experiment from among the hundreds stored at the IPA database or from an input file of their own. VitaPad allows the display of expression data for a single experiment or as a ratio for two experiments (test versus control). The user can enter search criteria to select an experiment and view detailed information on that
Visualization tools for the analysis of pathway data

Fig. 3. The database sub-schema for biological data. This is a fairly standard schema for metabolic pathways, and different tables would be needed for different types of pathways such as those encountered in pharmacokinetics.

experiment, including the procedure involved, microarray method (e.g. Affymetrix) and date of upload. Additionally, the user can customize how the data will appear by editing the gradient used to display the values. The gradient-edit menu shows the gradient used and offers a preview of how individual values will look in the specified color scheme. The user can modify the gradient by specifying the range of possible values and the colors used for the maximum value, minimum value, null value and, if multi-experiment, the value for 1. They can also save a custom gradient for future use or load a pre-existing one.

SYSTEM ARCHITECTURE

There are two components to the VitaPad database schema: information related to rendering and information related to data of biological interest. The data sub-schema (Fig. 3) is fairly standard, and is therefore discussed briefly. It contains various tables that record data on various classes of pathway-related entities—gene, enzyme, reactions, compounds, and details of gene expression experiments (Expression_Header and Expression_Detail). There are also 'bridge' tables that associate these classes—for example, the table reaction-compound notes which compounds participate in a reaction, and for each compound in the reaction, whether it is a substrate or a product. Additional tables may be added to represent new categories of data if desired.

The challenge in building an extensible pathway database is how to integrate new classes of data with the pathway-rendering process without having to modify the source code in a major way. (While we are distributing VitaPad as open-source to facilitate its modification for unforeseen eventualities, it is desirable to create a design that can accommodate most changes without any code alterations at all.) We address this problem through the rendering sub-schema, illustrated in Figure 4. This uses a modification of an approach originally devised by Tom Slezak and co-workers in the context of informatics support for the chromosome 19 mapping project at Lawrence Livermore in the early 1990s (Slezak et al., 1995): it has since been adopted by a variety of systems, notably the NCBI databases, the Human Genome Database (Letovsky et al., 1998) as well as the EAV/CR (Entity-Attribute-Value with Classes and Relationships) data model for bioscience data (Nadkarni et al., 1999; Marenco et al., 2003). EAV/CR is a data modeling approach well suited to rapidly changing database schemas, and seeks to address the problem of maintaining the user interface to the database through extensive developer-defined metadata on how a class of data and its attributes are to be presented to the user: this metadata is used for dynamic user-interface generation. A summary of EAV/CR is available at http://www.ycmi.med.yale.edu/nadkarni/EAV_CR_frame.htm:

Every class of 'atomic' object that will be rendered on a pathway diagram (e.g., gene, enzyme, compound, reaction) has a record in the Class table, shown in the top left of Figure 4. The fields in the Class table are:

1. A sequentially machine-generated Class_ID.
2. Class Name: The caption that is displayed to the user, e.g., 'Enzyme'.
3. A Table Name: This is the physical table (from the data sub-schema) where details of a particular object are stored for a given class. This is usually the same as the class name, but may not always be so (e.g., for the physical table 'Expression_Detail', we use the caption 'Gene Expression Value').
4. Primary Key Field Name: The name of the primary key field in the table corresponding to Table Name. Both these fields together allow browsing of the details of an individual object in the diagram (which is associated with an ID that is unique across the database) when that object is selected. (In terms of database access, the SQL code that is generated is 'select * from Table_Name where Primary-Key-Field_Name= object_id'.)
(5) Drawing_Object_Type: This field provides a broad indication of how the class will be rendered. It takes one of the following values: vertex, edge, edge-decoration.

(6) Every object that belongs to any of the Classes that are drawn has an entry in the Object table. This table acts as a central dictionary for every object in the database: the Object_ID field in this table takes the value of a primary key in one of the many class tables of Figure 3, while the Class_ID column records the class it belongs to. The advantage of a central object dictionary is that one can add associated tables of data such as synonyms. This enables implementation of a Google-like search-capability, where one can locate any object in the database by keyword, and get at its details, irrespective of which class it belongs to.

(7) The Pathway_Class table has the primary key (Pathway_ID, Class_ID). It allows one to override the default Drawing Object Type for a particular class of object for a given pathway. For example, while the designation of genes as edge-decorations may be appropriate in most cases, a user may wish to emphasize the genes in a given pathway and want them to be rendered as vertices instead.

(8) The Pathway_Appearance table has the primary key (Pathway_ID, Class_ID) and records the default presentation style attributes for each class of data. The ‘shape’ field currently takes one of the values: rectangle, ellipse and rounded-rectangle. The other fields in it are self-explanatory. One should note that certain attributes, such as fill-color, apply only to vertices and edge-decorations, but not to edges. A specially designated (‘default’) Pathway ID is used to store the default ‘global’ styles for the VitaPad installation. The information recorded for the default pathway is used to dynamically customize the appearance of icons corresponding to individual classes in the drawing palette.

(9) The Pathway_Vertices table records the objects that are the vertices for a given pathway. For each such object, we record its X and Y positions (in pixels). Note that while these are typically generated automatically through the GraphViz package, the positions may have been customized by a user, and must be saved for later reuse.

(10) The Pathway_Edges table records objects that happen to be edges in the graph. For each such edge, we record the object IDs of the two vertices that it connects to, as well as the edge’s curvature and direction. (In metabolic pathways, edges correspond with reactions, and direction information is used to draw an arrowhead indicating the direction of the reaction.)

(11) The Pathway_Edge_Decoration lists all the objects associated with a given edge in a pathway. For each decoration, we record a serial number, which indicates the order in which it is drawn on an edge. Objects with a smaller serial number are drawn closer to the start of the edge.

Fig. 4. The database pathway-rendering sub-schema. The Pathway table is shared between this sub-schema and the sub-schema of Figure 3. For a detailed explanation of the purposes of the other tables shown here, please refer to the text.
The Pathway_Object_Appearance table is used to allow the user to customize the appearance of specific objects in a pathway. This may be done, for example, to highlight the appearance of a critical gene or molecule. The structure is very similar to that of Pathway_Appearance.

Supporting the display of a new class of objects that has been added to the data schema essentially involves two steps. The first is to make an entry for it in the Class table, so as to specify how an object in that class is to be browsed, and to indicate its broad shape category. The next is to create an entry for it in the Pathway_Appearance table for the 'default' pathway, which specifies more detailed style information. The display process uses a straightforward inheritance mechanism, where the style information for individual objects supersedes that for an individual pathway, which in turn supersedes that for the default pathway. This means that if no details are specified at the first two levels, the default style information will be utilized.

CURRENT STATUS, LIMITATIONS AND FUTURE DIRECTIONS

VitaPad is currently in limited use within the Zhao lab and its research collaborators at Yale. By making it open-source, we hope to get feedback that will increase its utility to a wider range of researchers who are working on varied pathway-related problems.

Our current choice of Graphviz as the layout-rendering engine is not free of problems. In general, when laying out a two-dimensional graph optimally, one can attempt to minimize the relative spacing between vertices, or minimize the number of crossings between edges (or between edges or vertices when the latter are rendered as shapes). One cannot simultaneously minimize both of these parameters. Graphviz favors the first optimization, but does not provide options for the second. In particular, it provides little or no control over edge layout: edges are simply rendered as splines, which approximate to straight lines in some circumstances.

We work around this limitation currently through a manual approach. Specifically, we allow the user to treat a particular graph edge, if desired, as a series of straight line segments, which are connected by intermediate ('dummy') vertices designated by the user, whose position can be manually adjusted. This allows minimization of the number of edge-edge or edge-vertex crossings by having one edge ‘bypass’ another edge (or vertex) that it would otherwise cross.

To minimize the necessity for manual intervention, however, we are exploring an alternative layout engine, aiSee, which is a commercialization of Georg Sander’s VCG ('Visualization of Compiler Graphs') (Sander, 1995), which is available for a very modest academic price as well as registered at no cost for non-commercial use. (VCG is used in Pathfinder, cited above.) The aiSee software has numerous options for crossing optimization.

To enhance usability, we intend to incorporate prefuse, a toolkit for displaying complex interactive visual data developed at UC Berkeley (http://jheer.org). By utilizing an extensible design framework as well as a state-of-the-art rendering library, prefuse would provide a considerable amount of dynamic control over the layout algorithm and allow for a wealth of user-friendly interaction controls, including zooming, fisheye scoping and animation.

One imminent enhancement in VitaPad is the addition of a new drawing object type, the sub-graph. Sub-graphs allow different parts of the overall graph to be co-located. One use of sub-graphs is to indicate anatomic location. For example, in a pathway diagram that combines information about glycolysis (which occurs in the cytosol) with the citric acid cycle (which occurs in the mitochondria), it is useful to partition the graph into two sub-graphs. These may be visually indicated as boxes that enclose sets of edges/edges/decorations, accompanied by labels that indicate the specific sub-cellular location where specific reactions occur. Similarly, for pathways involving drug pharmacokinetics—absorption, distribution, action at target/s, biotransformation and excretion—sub-graphs indicating the various organ systems where specific processes occur are essential.

From the database perspective, support for sub-graphs implies allowing pathways to be components of other higher-order pathways, and allowing certain objects to be members of more than one pathway, so that they act as connectors between sub-graphs. For example, in the combined glycolysis–citric acid cycle pathway diagram, the connector role is played by pyruvate ion, which is transported into the mitochondria from the cytosol. This also means invoking GraphViz in two passes—initially, to arrange the objects in each sub-graph, and finally, to lay out the overall graph (collapsing the sub-graphs to point objects).

In future versions of VitaPad we expect to more fully incorporate the EA/CR architecture mentioned earlier, which allows creation of highly detailed specifications that support sophisticated display, browsing and editing of data through dynamically generated interfaces, as well as robust ad hoc query that utilises knowledge of the structure of classes and inter-class relationships.

The current XML format for data export/import is an amalgam of the Pathway_Objects_Appearance, Pathway_Vertices and Pathway_Edges tables: the style of every object on the pathway is specified, as are the X and Y coordinates of every vertex. The scope of this format is currently limited to display purposes only: the details of the biological data relating to individual classes of objects (which may contain fields that are often specific to the needs of a particular investigator) are not addressed. As the BioPAX specification matures, we expect to eventually support data interchange between VitaPad data and other packages that support BioPAX. As part of our incorporation of the EA/CR framework, however, we expect to support EDSP (EA/CR DataSet Protocol), an XML-based format that has the advantage of being schema-independent. Described in Marenco et al. (2003), EDSP achieves its independence by dividing the XML stream into two parts: one that describes the metadata (the classes, attributes and inter-class relationships within a schema), followed by the data proper in the form of attribute-value pairs associated with each object.

In summary, VitaPad offers a variety of unique features that enhance a researcher’s ability to generate pathway diagrams and display microarray data. Furthermore, the program’s open and extensible framework ensures that it will be useful in a wide variety of settings and capable of dealing with future concerns of genomic research.

Availability of Software: Source code and binaries are available at http://bioinformatics.med.yale.edu. The source code is also accompanied by limited documentation and a Microsoft Access database containing the schema described in the paper that is a clone of the IPA schema. This database contains the ‘public’ data from IPA, which has been created from the contents of public databases such as KEGG. The contents of the tables in this database may be readily exported to other databases (e.g., as tab-delimited text) using Access’s built-in export capabilities.
ACKNOWLEDGEMENTS

We thank two reviewers for their helpful comments. This work was supported in part by NIH grants GM 59507 and ES10867 and NSF grant DMS 0241160.

REFERENCES


