Partial Automation of Database Processing of Simulation Outputs from L-systems Models of Plant Morphogenesis

Yi-Ping Phoebe Chen\textsuperscript{1} and Jim Hanan\textsuperscript{2}

\textsuperscript{1}Cooperative Information Systems Research Centre, Queensland University of Technology, QLD 4001, Australia
email: p.chen@qut.edu.au

and

\textsuperscript{2}Centre of Plant Architecture Informatics, University of Queensland, QLD 4072, Australia
email: jim@csee.uq.edu.au

Summary

Models of plant architecture allow us to explore how genotype environment interactions effect the development of plant phenotypes. Such models generate masses of data organised in complex hierarchies. This paper presents a generic system for creating and automatically populating a relational database from data generated by the widely-used L-system approach to modelling plant morphogenesis. Techniques from compiler technology are applied to generate attributes (new fields) in the database to simplify query development for the recursively-structured branching relationship. Use of biological terminology in an interactive query builder contributes towards making the system biologist-friendly.

Corresponding Author: Yi-Ping Phoebe Chen

Keywords: L-Systems, Plant Architecture, Data Modelling
1. Introduction

Developmental processes in plants are controlled by many genes. For instance, in the garden pea (*Pisum sativum* L.), six key genes have been found that control branching, while another eight to ten genes control flowering. The genes operate through a variety of control pathways, influenced by the plant's environment and internal state. Recent developments in modelling plant morphogenesis (Prusinkiewicz 1998 & Prusinkiewicz et al 2000a) have made it possible to simulate these control processes to aid in unravelling the complex, dynamic system that is the growing plant.

Simulations are run using a variety of environmental conditions, with many replications to account for variation. Each plant may be made up of many components, such as internodes, leaves and buds, each with multiple parameters to characterise them. These components are arranged in a multi-level branching pattern, rendered irregular by interaction with the environment. Analysis of the developmental process requires that the parameters be recorded at a number of times during the life of the simulated plant. This yields very large quantities of data, even for a single plant and may take a long time, even on modern supercomputers. Exploration of a parameter space of this size is a huge task. Therefore, as many data as possible should be extracted from the simulation runs, and kept in an easily accessible, persistent format. The same data modelling and persistent storage issues arise in many other applications of dynamic plant modelling, particularly in plant architectural ecology where interactions among many plants may be involved.

The same data storage issues arise when collecting experimental plant architecture data. The hierarchical nature of the information can be captured in flat files by using computer-aided coding of structure (Hanan and Room 1997, Room et. al 1996) or using a special purpose database built around a multi-scale model of plant architecture as in the AMAPmod statistical plant modelling system (Godin et al. 1995, 1999).

In other areas with similar data handling problems, scientists often use commercial relational database management systems. For example, molecular biology databases, such as the Genome
Data Base at Johns Hopkins School of Medicine, Baltimore, the Genome Sequence Data Base at the National Center for Genome Resource and the Mouse Genome Database at Jackson Laboratory, Maine, have been developed using the Sybase relational database management system, which has adequate database management facilities for these tasks (Chen and Markowitz, 1995). In other cases, where the commercial products did not supply adequate facilities, research groups have developed their own tools. The Lawrence Berkeley Laboratory (Chen, et al, 1998) developed the Object-Protocol Model to make queries in terms of objects and protocols specific to their molecular biology research. The European Bioinformatics Institute (Park et al., 2000) developed representative sequence databases for efficient homology searching. Our goal is to adapt commercial products to our plant modelling problem, so as to minimise investment in tool development.

This paper describes a generic approach to data modelling and persistent storage for use with the widely used L-system formalism for modelling plant morphogenesis. The L-system rules that capture the development of plant architecture are processed to provide a description (schema) of an appropriate database structure. At the same time, an input routine is derived that can process the simulated structure at selected time steps to populate the database. This gives the researcher access to all the power of relational databases for analysis of the simulation data. In addition, special processing of hierarchical branching structures is performed to allow use of standard query languages in this analysis. To make the system easy to use, biological terminology is incorporated into a domain-specific query builder.

2. L-Systems

Plants are composed of building blocks (modules) such as leaves, flowers and internodes (pieces of stem) arranged in patterns controlled by genetics and environmental influences (Room
et al. 1994). Recognition of this modular nature of plants allows the modelling of plant architecture by representing each type of component as a symbol with associated parameters describing its current state (Prusinkiewicz and Lindenmayer 1990). A sequence or string of these symbols then captures the relative positions of the components. A branching architecture is imposed by enclosing the symbols representing a branch in brackets [ ] located between the two symbols adjacent to its point of attachment. Each successively nested pair of brackets describes another order of branching. Geometry is incorporated by including symbols that specify relative arrangement from one component to the next.

In order to capture development of the plant, an L-system (Lindenmayer 1968) takes this representation and adds production rules to specify how the components develop and grow. Each production rule takes the form

\[
\text{predecessor: condition} \rightarrow \text{successor}
\]

where \textbf{predecessor} is the symbol and parameters to be transformed, the condition must evaluate as true for the production to be applied, and the \textbf{successor} is the string that will replace the matching predecessor. Starting with an initial string called an axiom, every symbol in the string has the appropriate rule applied, producing a new string that represents the next stage in the plant's life. As an example consider the following:

Axiom: A(1)

Productions:
\[ A(n) \rightarrow I[L(n)][B]A(n+1) \]

where

- A represents an apical meristem or growing point in the plant, and its parameter value is the number of the node it will next produce,
I represents an internode, or piece of stem between two leaves,

B represents a lateral bud,

the brackets [ ] enclose a side branch such as a leaf or bud,

and L represents a leaf with parameter value its node of attachment.

Graphical commands have been left out for clarity. Each internode I will be drawn as a cylinder and each leaf L will be drawn with the surface shape appropriate for its position along the stem.

This would produce the following set of strings (Figure 1):

Axiom: A(1)

Step 1: I[L(1)][B]A(2)

Step 2: I[L(1)] [B]I[L(2)] [B]A(3)


...
The software used for this part of the L-system was developed by Prof. Prusinkiewicz’ research team at the University of Calgary (Prusinkiewicz et al., 2000b). For practical research applications, this is complemented by a process for measuring plant structure at intervals, deriving statistical characteristics for development and growth, and modelling plant ecology as a set of growth rules expressed in the L-systems formalism (Hanan and Room 1997).

3. Methods

3.1. The Database Representation of Branching Structure

Information about the components making up a plant could just be captured in flat files for all component types. This could include spatial information such as location in 3D space, but the relationships defined by the connectivity of the plant components would be lost. The position of a component within the branching hierarchy is of central interest in studies of plant architectural ecology. The modular nature of plants that lends itself so well to modelling using L-systems also provides a straightforward means of representing any branching structure in a database.

Figure 2 shows a simplified Entity Relationship (ER) conceptual schema that captures this. Internodes are the backbone of the branching hierarchy, defining the locations of child branches and of other components such as leaves and flowers. The sequence of internodes produced by a single apical meristem can be classified as a shoot or branch. Each branch and component is given an arbitrary unique identifier (ID) so that it can be identified within the database. Our approach is to assign the ID sequentially from the base of the plant traversing the branching hierarchy in a depth-first order. Since the ordering of internodes along a stem from the bottom up corresponds to their time of origin, oldest to youngest, and positional aspects of component properties may be important, each internode is assigned a sequence number from the beginning of the branch. In
conjunction with the branch ID this serves as the internode ID. Each component, such as a branch, leaf, or flower, can be identified as belonging to an internode within the hierarchy, typically the internode below its point of attachment. Other attributes such as plant and component age would be included in the appropriate table.

Each plant is made up of a number of branches, so there is a one to many (1:N) "Consists of" relationship between plants and branches. The hierarchical nature of the branching structure, where a branch may be the parent of many child branches, results in the 1:N "Parent" relationship from the branch table to itself (in database terms this is considered a recursive relationship). Each branch (except those originating at ground level, in particular the main stem in many plants) will have an internode at its origin. Since more than one branch may appear at a node, there is a 1:N "Origin of" relationship between internodes and branches. Since each branch is made up of one or more internodes, there is a 1:N "Consists of" relationship between branches and internodes. Since each internode may have an arbitrary number of components attached, there is a 1:N "Attached" relationship between internodes and components.

In Lindenmayer systems the bracketed strings produced by running the models are interpreted as branching structures. Particularly, a left bracket appears to the right of the internode in the parent branch which is the "Origin of" the child branch, while the matching right bracket terminates branch specification. The branch "Consists of" all internodes appearing between the left and right bracket that are not enclosed in another set of brackets. If internodes do appear in nested brackets, this indicates higher-order branches, and defines the "Parent" relationship. If more than one branch occurs at a site, the bracketed sub-strings representing each one are listed sequentially. Components, which may also be enclosed in brackets, appear between the internode which they are "Attached to" and the next internode in the current branch.
3.2. From L-system to Database

The central problem we address is to transfer the information on plant structure contained in an L-system string into a relational database. The basic architecture of the database management tools developed for L-systems (L-DBM) is shown in Figure 3. The visualisation of the virtual plant is
created by the University of Calgary L-system Interpreter, taking L-system productions as input and producing L-system strings.

The L-DBM consists of:

- a generator (L-DBM: Schema Translator) that derives an appropriate database schema from a set of L-system productions, and generates an interpreter (L-DBM: Population Interpreter) that can be used to populate the database automatically from the bracketed strings produced by the associated L-system.

- a domain-specific query builder (L-DBM: Query Builder) designed to handle the recursive database structure representing the branching structure of a plant.

The L-DBM takes two input files (top of Figure 3): one is the L-system declaration, the other is the L-system productions. The L-system declarations have to be added by the user to describe the types of components used in the L-system grammar. The L-DBM schema translator will automatically produce a database schema and an L-DBM Population Interpreter for this particular L-system. The input of the L-DBM Population Interpreter is a file of bracketed strings produced by running the same L-system. The output of the interpreter is the population of the database system that the L-DBM schema translator has specified. The L-DBM has been implemented using C and scanner and parser tools from the GMD tool box: Rex, Lalr, Cg and Puma (Grosch, 1992).
3.3. Using the L-Systems Grammar to Derive the Relational Schema

The basic schema that describes a branching structure must be given a concrete form before it can be populated for each specific L-system simulation. The L-system declaration and grammar are used to do this. An L-system declaration defines which symbol is to represent the internode (the key structural element), and provides user-friendly labels for other component tables.

The declaration also specifies the form of a module added to every L-system string to identify individual plants. Since this module may only appear in the L-system axiom and not in the
predecessor of any production, parameter names of any desired attributes must be included in the declaration. Plant ID is required to distinguish between individuals in the database, age may be required to distinguish between the same plant at different stages in its development, while other attributes can be used to distinguish between simulated experimental treatments. For instance, if we run a simulation with 2 environment parameters, temperature and photoperiod, each having 3 levels, and we have 10 stochastically different plants for each of 3 different genetic combinations and we want to analyse plant data at daily intervals for 6 weeks, we would have 10 plants x 42 days x 3 genetic types x 3 temperatures x 3 photoperiods. We could record the specific information for a particular plant in the axiom, for instance with the module PLANT(ID, Age, GeneType, Temperature, Photoperiod).

We use the L-systems productions to derive the specific schema for the branching structures that the productions will simulate. Figure 4 shows an Entity Relationship conceptual schema describing an example L-system. Components are associated with the internode appearing in the production that produces a module that leads to their development. Each component table has its attributes defined by the parameter names appearing in the predecessor of the first production for the corresponding symbol. Tables will only be defined for symbols listed in the declaration. The presence of a branch is identified by a module occurring within [ ] in the right hand side of a production having an internode symbol appearing in its own expansion. This will produce a recursive structure in the database (enclosed in a dashed line in the figure). It will also indicate the position of brackets that will cause a new Branch ID to be assigned, and the sequence of Internode IDs to be initialised to zero, when populating the database.
In our simple example, the most important and interesting part of the schema in Figure 4 is inside the dashed line, where the schema is produced by production rules P1 and P2. From the first production P1: A \rightarrow I[L(0)][B]A, we find that "A" produces a branch since it has an internode I in its right hand side, but it does not directly define a branching relationship itself since the "A" is not enclosed in brackets. However, the symbol "B" in the right hand side does have "[ ]" around it, and the expansion P2: B \rightarrow A will be followed by the replacement of "A" by "I[L(0)][B]A" through production P1, and therefore defines a branching relationship. The presence of the internode in production P1 also defines the connection relationships between components and internodes. The age attribute for the leaf component is derived from the parameter name appearing in the predecessor of production P3.
(1) P(1,0,1,20,6)A
(2) P(1,1,1,20,6)[L(0)][B]A
(3) P(1,2,1,20,6)[L(0)][B]A
(4) P(1,3,1,20,6)[L(0)][B]A

FIGURE 5. Four strings (steps 1 to 4) for the example L-system

As an example of multiple recursive structures in a schema, we add production P4 defining a root (Figure 6). The Shoot and Root entities belong to the same plant but are not related directly to each other since they do not share any symbols. The top branch (Shoot) structure has its own recursive structure and attributes, as does the bottom structure (Root).

L-Systems Declaration
A - Bud
B - Bud
I - Shootinternode
L-Leaf
R-Root
J - Rootinternode

L-Systems Productions
P1. A --> I[L][B]A
P2. B --> A
P3. L(age) --> L(age+1)
P4. R --> J[R][R]

FIGURE 6. Two Recursive Database Structures (Shoot and Root) in one Schema
3.4. Translating L-systems strings to Database Population

At the same time that the L-DBM Schema Translator produces the schema for an L-system it will generate a population interpreter to accept strings produced by the L-system. Different L-system declarations and productions will produce different population interpreters. Techniques and tools from compiler (Pittman and Peters, 1992) technology are used in this process. The data flow of the Population interpreter is shown in Figure 7.

The Scanner reads an input file of L-system strings and recognises from it brackets and the symbols (called tokens) specified by the user in the L-system Declaration. The tokens are passed from the Scanner to the Parser, which then builds an intermediate data structure called an abstract syntax tree. The Constrainer analyses the abstract syntax tree and attributes evaluation rules to associate components with the appropriate internode and branch and to calculate derived attributes required to position and uniquely identify internodes and components within the plant. These were identified during schema translation by analysing the production rules to determine which component symbols generate a sequence of internodes making up a branch and which initiate a new level of branching. For each set of internodes making up a branch, sequence numbers will be assigned. At each new level of branching, depth within the structure is calculated, which will allow easier querying of the recursive branching relationship. Unique identifiers are generated for each new branch and for each new component. Finally, the Code Generator outputs database transactions to populate the tables defined by the Schema Translator.
These tables show the information corresponding to the four strings. The branch table shows Branch_ID, this branch’s Parent_Branch, Origin_Internode and its related attributes, for example Branch Order. The leaf table needs Branch_ID as well as Internode_ID, since together they make the unique id of an internode. The leaf table also includes its related attributes, for example L_Age.
The derived attributes make querying the recursive schema using existing database techniques much easier.

**TABLE 1. Example Plant Table**

<table>
<thead>
<tr>
<th>Plant_ID</th>
<th>Age</th>
<th>GeneType</th>
<th>Temperature</th>
<th>Photoperiod</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>20</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>20</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>20</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>20</td>
<td>6</td>
</tr>
</tbody>
</table>

**TABLE 2. Example Branch Table**

<table>
<thead>
<tr>
<th>Branch_ID</th>
<th>Plant_ID</th>
<th>Age</th>
<th>Parent_Branch</th>
<th>Origin_Internode</th>
<th>Branch_Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>NULL</td>
<td>NULL</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>NULL</td>
<td>NULL</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>NULL</td>
<td>NULL</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

**TABLE 3. Example Internode Table**

<table>
<thead>
<tr>
<th>Internode_ID</th>
<th>Branch_ID</th>
<th>Plant_ID</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
TABLE 4. Example Leaf Table

<table>
<thead>
<tr>
<th>Leaf_ID</th>
<th>Branch_ID</th>
<th>Internode_ID</th>
<th>Plant_ID</th>
<th>Age</th>
<th>L_Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

4. The L-DBM Query Builder

The L-DBM Query Builder user interface is a step towards a biologist-friendly computing environment based on the ConQuer (Bloesch and Halpin, 1996) conceptual query language. When the user formulates their query they use an interface that looks like the window in Figure 9. The left hand pane of the L-DBM query tool is the query viewer, giving a dynamic representation of the users query as it is built. The user chooses the object they are interested in from the middle pane, drags it into the query viewer, then selects a relationship with another object from the right-hand pane. The L-DBM query builder supplies basic (such as +, -, *, /, =, <, >, >=, <= etc.), and aggregation (count, average, sum, etc.) functions. The user can drag these operations into the query builder to build queries in a suitable way. The list of objects and relationships includes precomputed structural attributes (such as Branch_Order, Parent_Branch, Origin_Internode) stored by the population interpreter as derived attributes (Chen and Colomb, 2000), simplifying queries related to the recursive relationships in the branching structure.
5. Conclusion

The complexities of plant architectural ecology can be addressed through application of biological informatics tools in the form of simulation coupled with the database technologies specially tuned for handling plant architecture. This research is at the intersection of two fast growing research areas; information technology and biotechnology. The capability to efficiently analyze measurement and simulation data is a central problem in this field. Sophisticated computer technologies are essential for managing and interpreting the simulation data. The research reported in this paper applies a database programming language to a specific domain L-system models applied to plant architecture research.
The L-DBM system enables persistent storage and ease of access to the masses of data that can be generated by L-system simulations of plant development. Once the L-system declaration has been provided, schema generation and database population are fully automated. The Schema Translator reads the declaration and productions for a particular L-system and produces the database schema and Population Interpreter. The Population Interpreter accepts any string produced by the associated L-system and produces the population of the schema produced by the Schema Translator.

The recursive nature of the database schema required to represent a branching structure makes querying difficult. One of key results of this paper is the use of the compiler code generator techniques to precompute standard recursively defined attributes. A query builder that represents the different data structures is also presented.

In future work in this area, we plan to extend the approach to incorporate generic techniques for capturing field measurements of branching structures. The research also intends to extend the biologist-friendly computing environment for analysis of plant architecture data, particularly in the area of difficult-to-query recursive database structures.

References


