Simulating Genetic Regulatory Networks

Richard Scheines & Joe Ramsey

November, 2001

Technical Report No. CMU-PHIL-124

Philosophy

Methodology

Logic

Carnegie Mellon

Pittsburgh, Pennsylvania 15213

Simulating Genetic Regulatory Networks

Richard Scheines and Joe Ramsey

1. The General Model

Consider a discrete time series involving N individuals, each with M variables V1 … Vm. For example, Figure 1 represents a time series for three variables, each measured at 5 times.

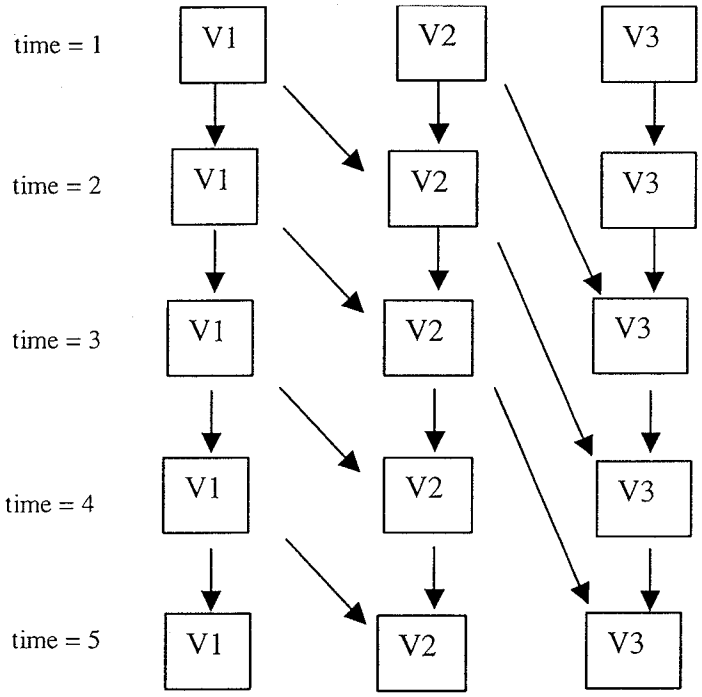

Figure 1: Causal Graph for 5 times

If we assume that the process modelled is stable over time, then we can represent the causal structure of the series with a **repeating graph** that includes the smallest fragment of the series that repeats. The number of temporal slices in the repeating graph is the longest lag of direct influence plus one. For example, the repeating graph in Figure 2,[1] which represents the series in Figure 1, needs three temporal slices to represent a repeating sequence, because V2 has a direct effect on V3 with a temporal lag of two.

[1] We connect all pairs of variables at the beginning of the repeating sequence with a double-headed arrow to represent an unconstrained causal connection, so that d-separation applied to this graph does not entail that these variables are independent, contrary to the fact that later versions of the same variable are causally connected in the repeating graph.

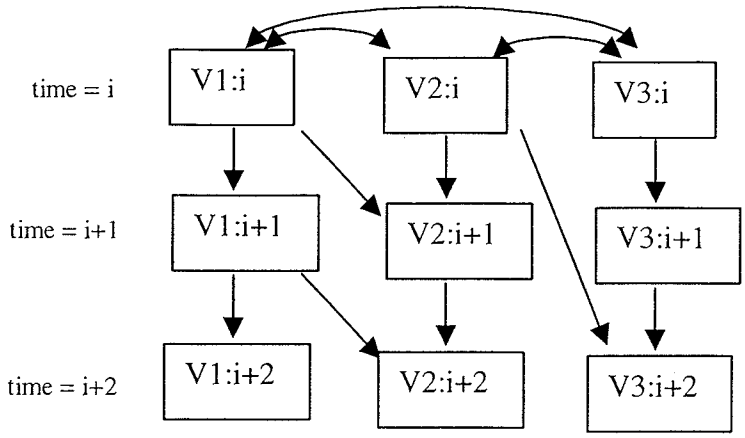

Figure 2: Repeating Graph

In order to simulate data from this class of models, we need to express each variable (or its probability distribution) at an arbitrary "current time" as a function of its direct causes. To represent the set of direct causes for each variable, we construct an **update graph.** For example, the update graph in Figure 3 expresses the causal information needed to simulate the time series represented in Figure 1 and Figure 2. The update graph consists of the M variables repeated in temporal slices from lag = mlag (most remote direct influence) to lag=0 (current time), but includes only edges that are into variables at lag = 0, that is, it includes *only edges that represent direct influences into the current time.*

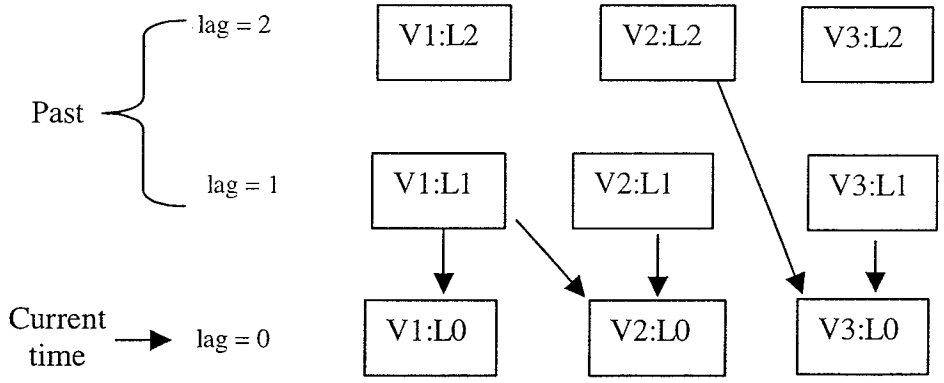

Figure 3: Update Graph

The time lag i in an update graph is indexed by ":Li". Thus, variable V1 at the current time (lag of 0) in an update graph is V1:L0, and the same variable two time slices in the past V1:L2. The constant *mlag* is the maximum lag of direct influence.

To simulate data in Tetrad 4, you must specify an update graph and then interpret it as a parametric model. There are 4 choices of parametric models, where in each either i) the value of each variable at the current time is a function of its causal parents and an error term, or ii) the probability distribution of the variable is a function of its causal parents. Given an instantiation of the chosen parametric model, and the assumption that the causal

relations remain constant over time for each individual, values for N individuals (cells) over T times can be simulated to produce a data cube that is:

N (individuals) x M (variables) x T (times).

Although the cube has three dimensions, we will store it in the standard two, by repeating M columns for each time slice, as so:

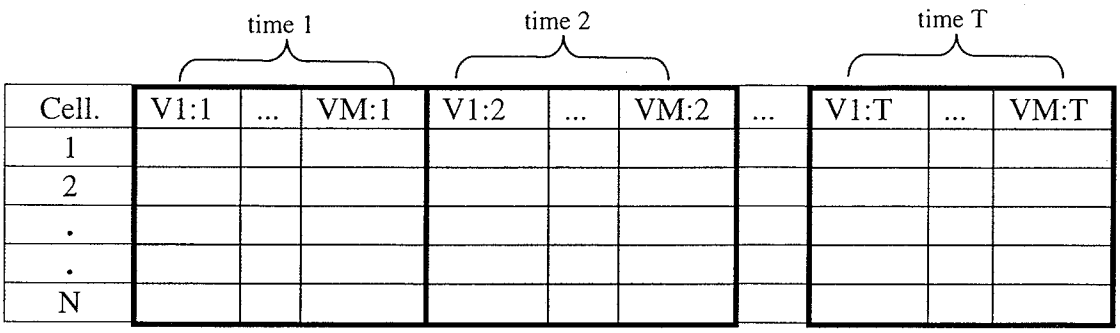

Figure 4: Data Cube

The simulation to produce these data is run in two stages: an "initialization" phase and an "update" phase. The initialization phase must assign values for each individual for at least as many times as the maximum lag in the time series model, i.e, from time t = 1 until time t = mlag. After time ≥ mlag, values can be assigned to variables for a given individual using the given instantiation of the parametric model that interprets the update graph.

Data collection regimes for protein expression involve tissues that contain thousands of cells, not all of which behave strictly identically and not all of which can be measured in isolation.

Since current technology cannot perfectly measure the levels, even relatively, of gene (or mRNA) expression, or levels of protein synthesis in cells, we also allow the user to specify a measurement model that aggregates cells by dish and models measurement error.

In what follows, we explain how to specify the graph, choose a parametric model to interpret the graph, instantiate the parameters of the model, pick an initialization routine, and finally specify the measurement model.

Starting the Program

The overall simulation specification begins by loading TETRAD-4, which is a Java application that can be downloaded from: http://www.phil.cmu.edu/tetrad/. The download will write a .jar file to your disk. 1) Make sure version 1.3 or higher of the

Java SDK (or JRE) is installed. If it is not - go to http://java.sun.com/j2se/1.3/jre/index.html to download it. 2) On a Windows machine, double clicking the jar file should start the program. If it doesn't, you need to modify the File Type mapping for the .jar extension so that the "open" action is "javaw -jar %1". If you don't know how to do this, either find someone who does, or take the command line option, which is: 3) In any case, the jar can be run on all machines (Linux included) by typing "java -jar <path-to-filename.jar>" at the command line.

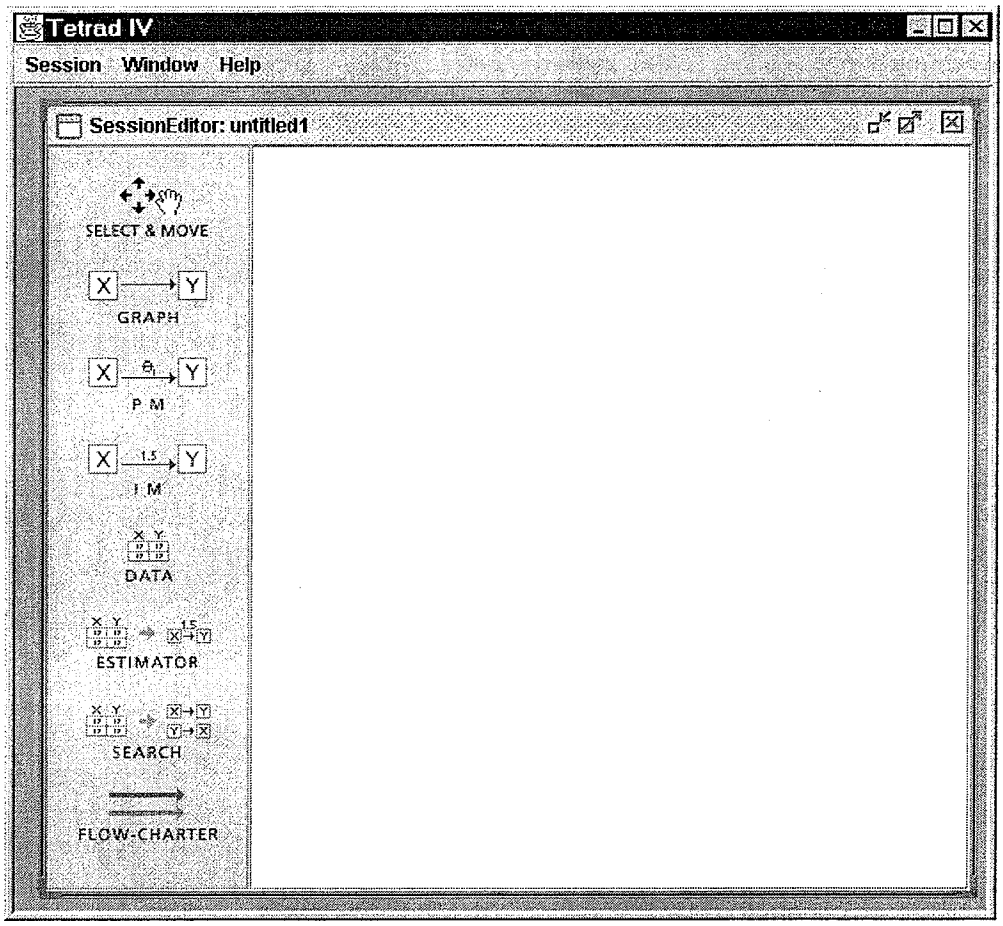

Figure 5: TETRAD 4 on Start-up

When the program opens, it will display the empty workbench we show in Figure 5. In order to generate data, you need to specify a graph, a parametric model (PM), an instantiation of this model (IM), and connect them all to a Data node. These objects can be deposited on the workbench by clicking on their icon in the left and clicking where you want them located on the workbench, and then by clicking on the "Flow-Charter" icon on the left and then dragging from one node to another to connect them. The skeleton for a simulation looks like Figure 6.

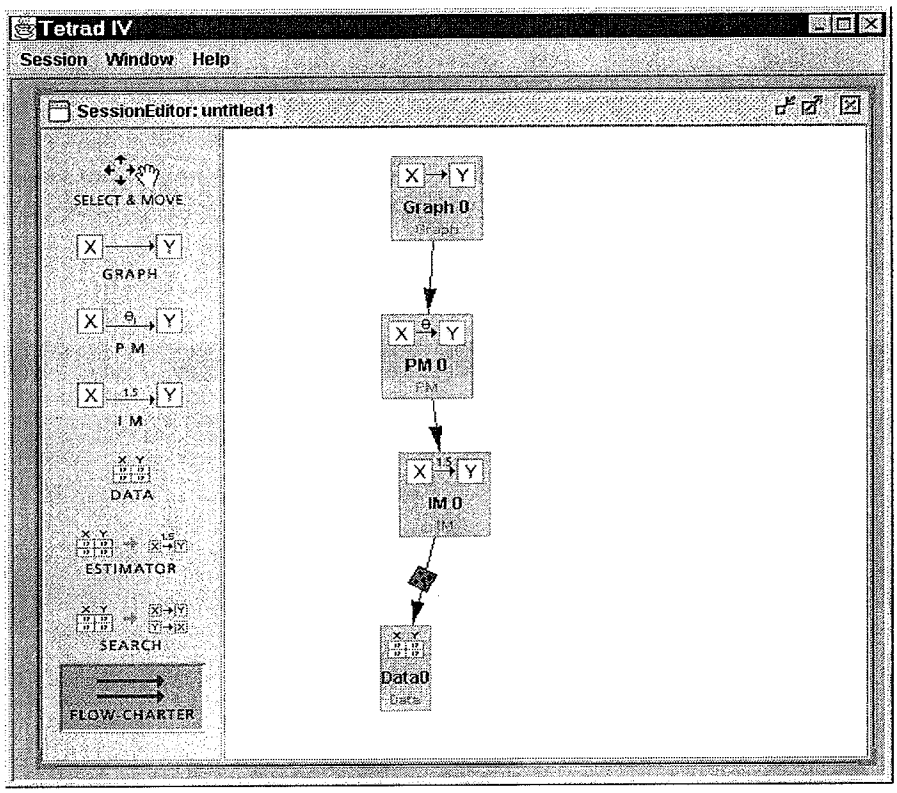

Figure 6: Simulation Session

To fully specify each piece of the simulation, you need to double-click on each node, beginning with the graph and moving down to the PM and then to the IM. We cover each in turn.

2. Specifying the Graph

After double-clicking on the graph node - you will be given a choice (Figure 7) of whether you want to specify a general graph or an update graph, with the default an update graph.

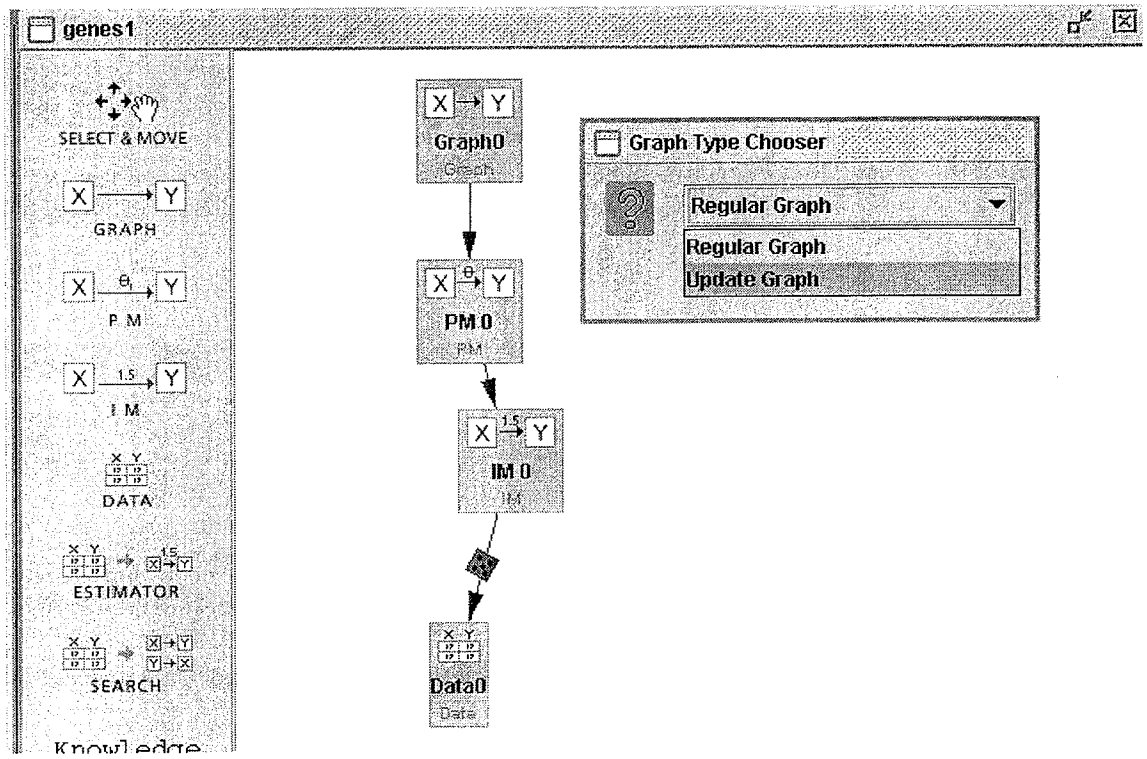

Figure 7

The Regular graph allows you to specify a Causal Graph interpretable as a Bayes Net or Structural Equation Model. An update graph allows you to specify a specialized structure for genetic simulations. Choose Update graph.

You will then be prompted to specify the graph manually or randomly, with randomly the default.

2.1 Manual Graph Specification

1) User Specifies *M*, the number of variables
(range: 1 to 500, default = 5)

2) User Specifies *mlag*
(range: 1 to 5, default = 1)

3) Default graph is drawn for user in graph drawing window with arrow from each Vi:L1 to Vi:L0, for mlag = 2 and M = 5:

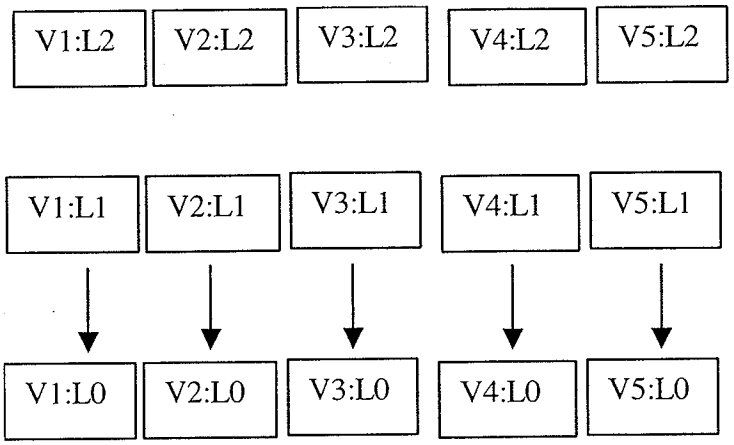

Figure 8

4) User completes graph manually, i.e., they add edges from variables at lag > 0 into variables at lag = 0. No other edges are allowed in this representation of the series.

2.2 Random Graph Specification

1) User Specifies *M*, the number of variables in each individual
(range: 1 to 500, default = 5)

2) User Specifies *mlag*
(range: 1 to 5, default = 1)

3) User chooses and sets the value of exactly one of:
a) constant indegree *ci* (range: 0 to M**mlag*, default = 2)
b) max indegree *mxi* (range: 0 to M**mlag*, default = 2)
c) mean indegree *mni* (range: 0 to M, default = 2)

a) *constant indegree* - choose parents(Vi) by putting a uniform distribution over the possible parents of Vi (i.e., all variables earlier in time) and drawing without replacement until |parents(Vi)| = *ci*

b) *max indegree* - for each variable in the set of possible parents of V_i ,include it in parents(Vi) if a random draw from a uniform [0,1] is greater than cutoff = 1/|possible parents of Vi|, until either the possible parents of Vi are exhaused, or |parents(Vi)| = *mxi* whichever is first.

c) *mean indegree* - for each variable Vi, decide for each variable in the set of possible parents of Vi to include it in parents(Vi) if a random draw from a uniform [0,1] is greater than cutoff = 1/|possible parents of Vi|, until the possible parents of Vi are exhaused.

3. Specifying the Parametric Model

You must now intrepret the graph as a parametric model. To begin this, double-click on the PM node in the session editor (Figure 6). When you are finished, just close the PM editor window.

If you have specified an update graph, then you will be given four choices (Figure 9) of parametric model families, each of which we describe below.

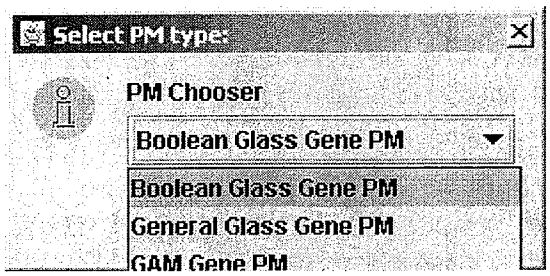

Figure 9

These families are not exclusive, but each has advantages.

3.1 Glass Updating

In both Glass and General Updating parametric models, the current value of each variable Vi:L0 is set by a function with the following general form.

Update Function: $\quad Vi{:}L0 = Vi{:}L1 + rate[-Vi{:}L1 + F_i(parents(Vi{:}L0)_{/Vi:L1})] + \varepsilon_i$

where: $0 < rate \leq 1$, and ε_i is an "error" term drawn from a given probability distribution (discussed below), all variables are continuous, and F_i is a function specified by the user. The difference between Glass and General updating models involves only the form of the functions F_i.

3.1.1 Preliminary Binary Projection

Even though the variables in this parametric model class are continuous, Glass functions take boolean valued inputs, so some pre-processing is necessary. The inputs to the function F_i are the members of the set $P = \{parents(Vi{:}L0)_{/Vi:L1}\}$, that is, the parents of Vi:L0 except for Vi:L1, which is already in the updating function. The idea behind Glass functions is to simplify the input to whether a given gene is expressing at "high" or "low" levels. We map each $Vp \in P$ to 0 (low, i.e., below its average expression level of 0) or 1 (high, i.e., above 0). We do this with the following binary projection **BP**:

BP: For each $Vp \in P$, let $Vp^* = 1$ if $Vp > 0$, and 0 otherwise.

Simulated values in for variables Vi will range mostly over the interval [-2.0, 2.0] and will oscillate above and below 0.0. Since raw microarray data is typically given as a ratio of microarray spot brightness to average spot intensity, where this ratio typically ranges from near 0.0 up to about 10.0, with averages around 1.0, it is useful to think of the simulated data as loglinear with respect to raw data—viz., log(x) of each intensity ratio x is recorded in the simulation in place of the intensity ratio x.

3.1.2 The Function Table for F_i

Since genetic regulators either inhibit or activate their targets, Edwards and Glass (2000, p.3) set the range of F_i to {-1,1}. To specify a particular F_i, construct a truth-table with 1 column for each $Vp \in P$ and one for the output of F_i. Thus the truth table will have $2^{|P|}$ rows. For example, if P = {V3:L1, V5:L2}, then the function table for F_i is:

V3:L1 *	V5:L2 *	F_i
0	0	
0	1	
1	0	
1	1	

Filling in the F_i column in this function table specifies the particular instantiation of F_i used in the update function for variable Vi, thus this step is left to the Instantiated Model specification below.

By picking the Glass Updating parametric family, you are committing yourself to the class of models parametrized by the update function above and Glass functions for the F_i.

3.2 General Updating

Again, in both Glass and General Updating parametric models, the current value of each variable Vi:0 is set by a function with the following general form.

Update Function: $Vi:L0 = Vi:L1 + rate[-Vi:L1 + F_i(parents(Vi:L0)_{/Vi:L1})] + \varepsilon_i$

where: $0 < rate \leq 1$, and ε_i is an "error" term drawn from a given probability distribution (discussed below), all variables are continuous, and F_i is a function specified by the user. The difference between Glass and General updating models involves only the form of the functions F_i. In General Updating parametric models, the user is free to specify any function for the F_i, not just Glass functions. Here we will use Tianjiao's GAM specifier to allow the user to specify, for each i, the function F_i.

3.3 GRN GAM

In this parametric family, the variables are continuous and the update function is slightly more general than the ones used in 3.1 and 3.2.

Update Function: $Vi:L0 = G_i[parents(Vi:L0)] + \varepsilon_i$

GAM stands for general additive model, so the only constraint on this parametric model is that the G_i are additive functions. The particular instantiaions of G_i for each i are fixed when you specify the instantiated model (IM). Here we will use Tianjiao's GAM specifier to allow the user to specify, for each i, the function G_i.

3.4 BN SEM

In this parametric family, the variables are discrete and the causal system is just interpreted as a standard discrete Bayes Network, i.e., the update function is just a conditional probability table.

Update Function: $P(Vi:L0) = G_i[parents(Vi:L0)]$

At the parametric level, you need to specify the number of categories for each variable, as well as the value of each category. This is identical to the Bayes Net parametric model in general TETRAD 4 models.

4.0 Specifying the Instantiated Model

Once the parametric model has been specified, you need only specfiy values for the parameters in the corresponding instantiated model (IM). To do this, double-click on the IM node in the session workbench. Close the IM editor to finish. Since the parameters depend on the PM chosen, we cover each of the above classes in turn.

4.1 Glass Updating

The free parameters of a Glass Updating model are:

- the rate
- the distributions over the error terms ε_i
- the Boolean functions F_i

The error distributions are all assumed to be pairwise indedpendent, and normal with mean 0 and standard deviation $SD(\varepsilon_i)$. The rate and error distributions default as follows.

Parameter Defaults
Rate = .1
$\varepsilon_i \sim N(0,.05)$

The range of F_i is {-1,1}. The number of possible functions for each F_i is $2^{2^{|P|}}$, where P is the set of parents of V_i. For example, if P = {V3:L1, V5:L2}, then the uninstantiated function table for F_i is:

V3:L1 *	V5:L2 *	F_i
0	0	
0	1	
1	0	
1	1	

which can be filled out in 16 different ways. If P contains 5 parents, then there are 32 rows in F_i and there are 2^{32} different ways you can instantiate F_i. Thus, we give you a choice of filling in all F_i randomly or manually, with randomly the default.

Randomly:

- Draw F_i from: ($p(F_i = 1) = .5$, and $p(F_i = -1) = .5$, but
- Ensure all inputs to F_i are *non-trivial*, i.e., for each input P_k, insist that at least one pair of rows in the function table exist that are i) identical on all inputs besides P_k, ii) different on P_k, and different on the output F_i.

Manually: Allow user to assign the value of F_i in each row manually.

4.2 General Updating Models

Same as Glass Updating, but use Tianjiao's GAM interface for specifying the functions G_i.

4.3 GRN GAM

Here we will use Tianjiao's GAM specifier to allow the user to specify, for each i, the function G_i, and specify the error distributions as above.

4.4 BN SEM

Same as TETRAD 4 Bayes Net instantiated model specifier.

5. Initializing the Cells

Given the full parametric model and its instantiation, and assuming that the same model is used for each individual cells, then values for N individuals over T times can be simulated in two stages: an "initialization" phase and an "update" phase. The initialization phase must assign values for all the variables (genes) in each individual for time step 1. After this, the update function specified above can be used to assign values to variables for a given individual, with the caveat that if mlag > 1, then not all the

"parents" of a variable in time 2 will exist. In that case the update function will simply use the latest value of that variable that does exist as input.

Biologically, cells from the same tissue can be starved of nutrients, or manipulated in some other way and all brought into roughly the same state, i.e., synchronized. They can then be shocked, e.g., exposed to nutrients, and let run.

Further, many genes in a cell are considered "housekeeping genes," which express protein at some basal rate stably over time. In this version of the simulator we allow the user to choose among two methods for initializing values: synchronized or random.

5.1 Synchronized Initialization

1. For each gene V_i in cell 1,
    a. if Vi is a housekeeping gene, i.e., its only parent in the update graph is itself, then let Vi:1 = 0,
    b. else draw Vi from a standard normal distribution, i.e, N(0,1).
2. Duplicate cell 1 N times, where N is the total number of individual cells.

5.2 Random Initialization

1. For each gene V_i in each cell,
    a. if Vi is a housekeeping gene, i.e., its only parent in the update graph is itself, then let Vi:1 = 0,
    b. else draw Vi:1 from a standard normal distribution, i.e, N(0,1)

6. Running a Simulation

To initiate a data generation process, double click on the red-die on the arrow from an IM node to a Data node (Figure 10). The program will only produce data if it has all the info it needs, otherwise it will prompt you.

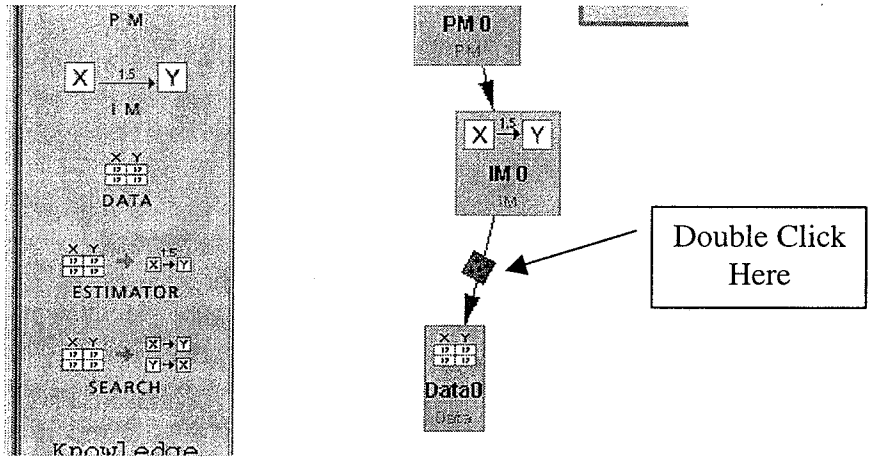

Figure 10: Starting a Simulation Run

The simulation will initialize the cells as specified above, and then generate as many time steps as desired. Not all time steps generated need to be stored, however. The updating process might happen at quite a different pace than data recording, and for inference this might matter. The simulator thus requires you to specify the first time step you want to store, and how often you want to store data.

Simulation Run Parameters:
Thus the parameters you must set, with their defaults in brackets, are as follows.

1. Number of individual cells = **N** [100,000]
2. Total time steps generated = **T** [4]
3. First time step *stored* = $\mathbf{TS_{start}}$ [1]
4. Interval for storage (step size) = **step** [1]

7. Aggregation and Measurement Error

7.1 Aggregation by Dish

So far, we have modelled the progression of gene activity over time for *individual* cells. We cannot currently obtain biological data on this level, however. In microarray experiments, thousands (or millions) of cells are grown in each of several dishes. For example, Figure 11 shows two dishes with a million cells each.

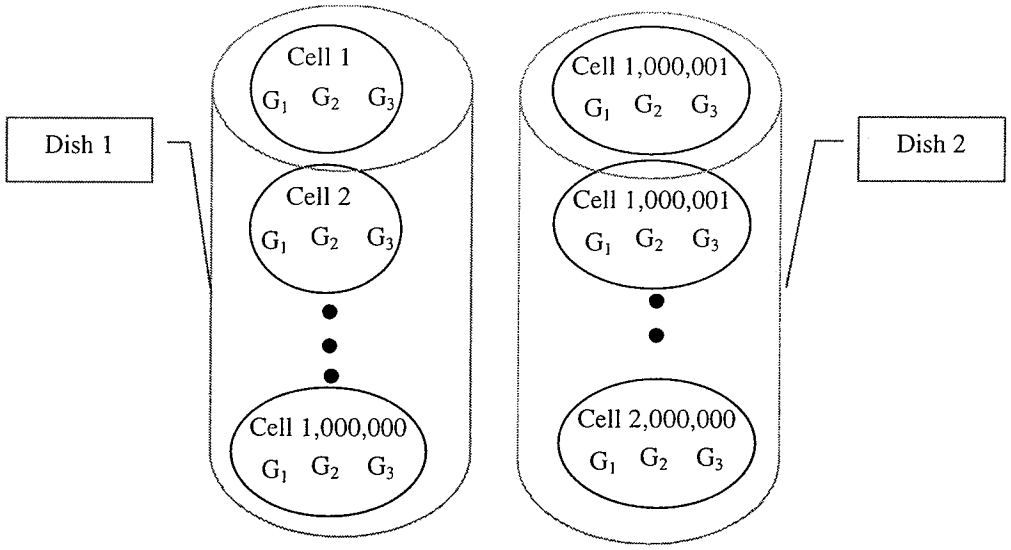

Figure 11: Aggregation by Dish

A microarray experiment involves at least the following sequence of steps to begin the experiment:

1. Separate single cell culture into dishes
2. Grow cells in dishes
3. Synchronize Cells
4. Expose dishes to treatment

Initial Dish to Dish Differences

Although they are intentionally minimized, small variations in nutrient or temperature make dish to dish variations inevitable, even at the beginning of an experiment.

To simulate these differences, we prompt the user for how much variation in expression levels we can expect between dishes (Figure 12). Let that quantity be the *standard deviation of expression level difference due to the dish*, which we will call sd(DV), with default at 10%.

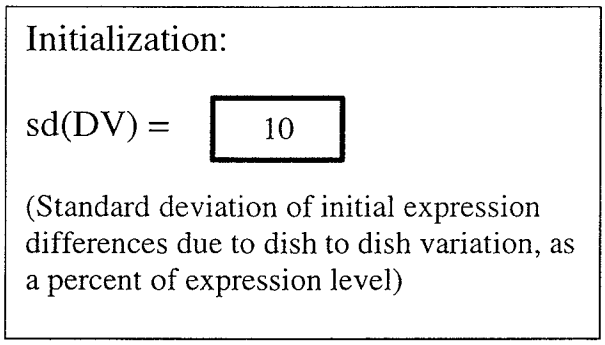

Figure 12: Dish to Dish Variation Parameter

For each dish D_j, we draw $Dev(D_j)$ from a normal distribution with mean 100 and standard deviation = sd(DV), i.e., N(100, sd(DV)).

We then adjust the *initial* [2] expression levels in all genes in all the cells in dish D_j by $Dev(D_j)$.

For each dish D_j
  For each cell C_k in Dj,
    For each gene G_i in C_k at time 1 (Vi:1),
      Let G_i = $Dev(D_j)$ % of G_i

Dishes and Time

In the usual studies, although several dishes might begin an experiment in a relatively "synchronized" state, each dish must be "frozen" before it can be processed for measurement, and thus we cannot use the same dish to measure cells at two different time points in the experiment. To measure two time points, we take a sample from one dish at time 1 and a sample from a different dish at time 2, e.g., Figure 13.

[2] We discussed how we initialize the cells in section 5 above.

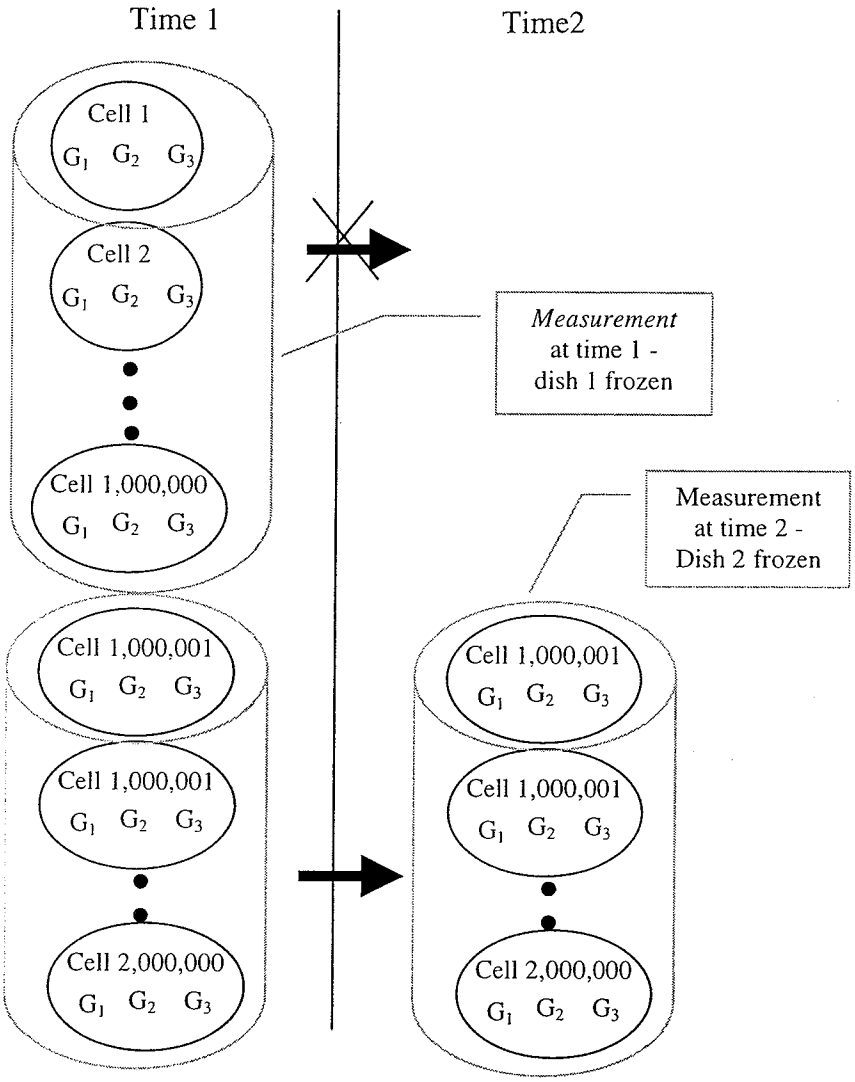

Figure 13: Time and Dishes

If the goal is to compare the expression levels of a particular gene at two different times to see if they substantially differ, then this constraint forces us to compare the level of a sample from one dish against a sample from another. This constraint will NOT be imposed by the simulator, but rather by the data analyst after data is generated and stored.

7.2 Measurement Error

After a dish is "frozen" at a time, its RNA is extracted. From this RNA, several samples can be drawn and each "measured" by exposing it to a microarray chip (chips can be re-used) and then digitally converting pixel intensities to gene expression levels (Figure 14).

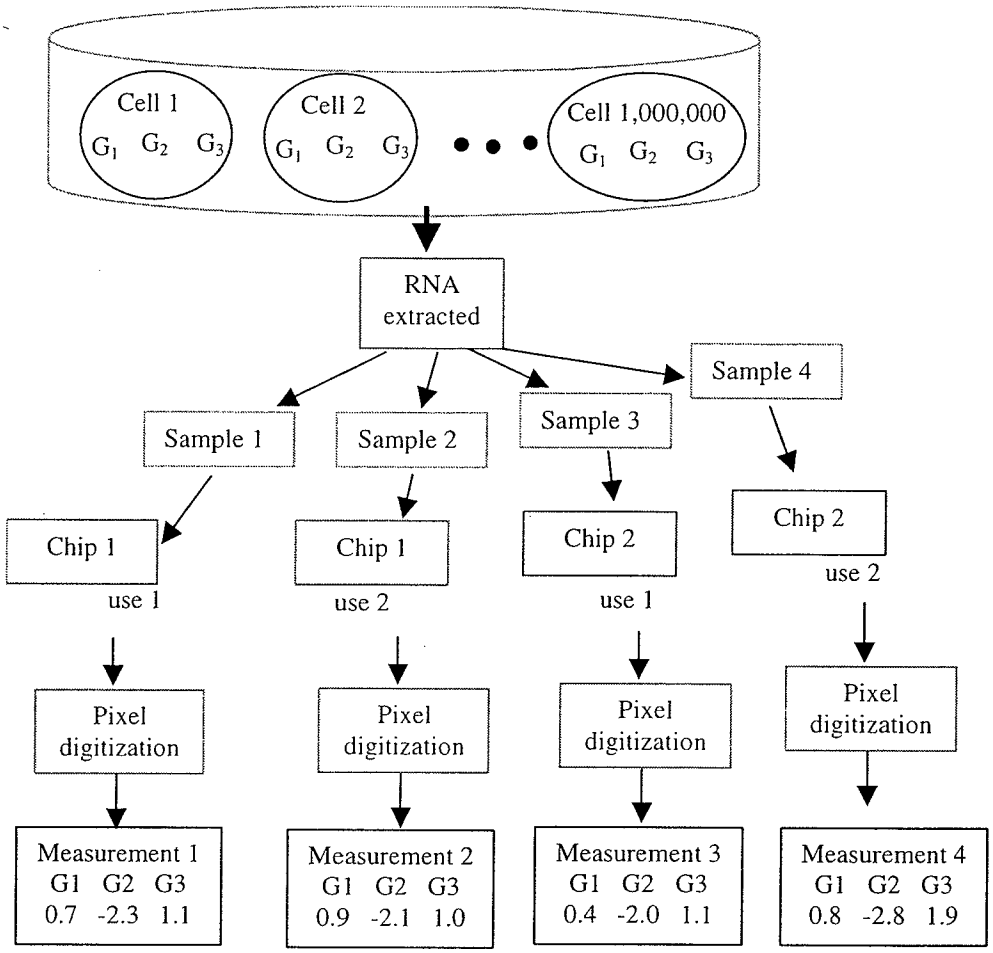

Figure 14: Measurement Error

Chips can be reused approximately four times. Samples from the same dish might vary slightly, chips definitely vary, the same chip functions differently from one use to another, and the pixel digitization routine might include error as well. Each "measurement" therefore, has five sources of potential error:

1. dish,
2. sample,
3. chip,
4. re-use of a chip, and
5. pixel digitization

We discussed how we will model the dish variability above. In this version of the simulator, we will NOT model dish to dish variability beyond initialization. To model the other sources of measurement error, we will proceed as follows.

After initializing the cells in an experiment, and adding dish to dish variabiltiy, we will have N cells partitioned into D dishes. After we run the experiment, in which each cell

obeys the same causal laws but updates independently of each other, each cell has an expression level at each of t times.

After Running the Experiment

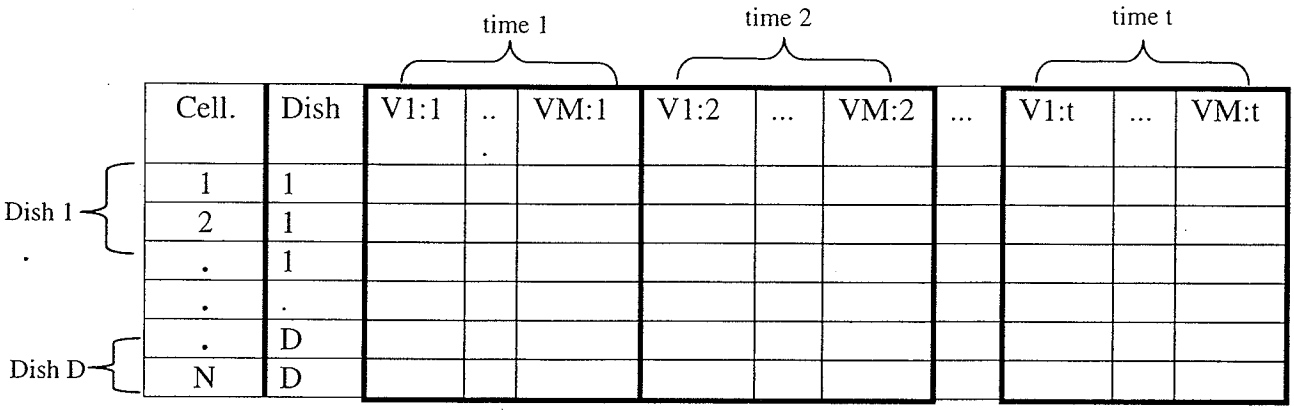

Figure 15: Gene levels before measurement error

After RNA Extraction

Next, we model the RNA extraction process for each dish by aggregating all the cells in a dish, averaging the expression levels for each gene, and recording an average expression level for each gene in each dish at each time (Figure 16):

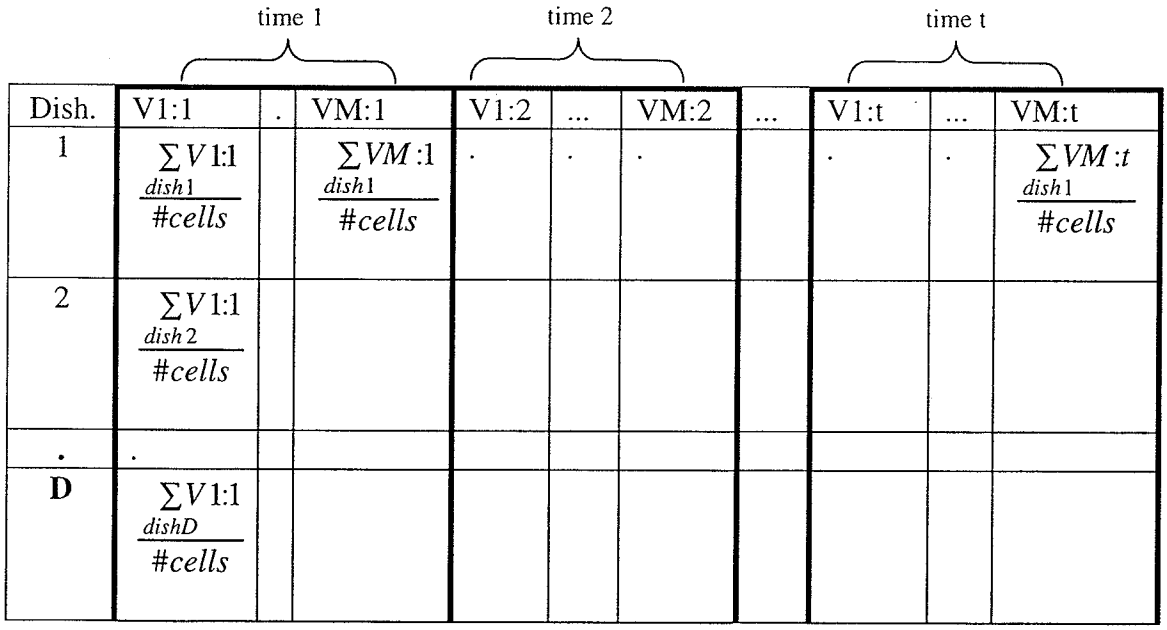

Figure 16: After RNA Extraction

The average expression level for a gene Vi at time t in a dish d, is

$$\text{Vi:t(d)} = \frac{\sum_{dish=d} Vi{:}t}{\#cells}$$

Adding Measurement Error

There remain 4 sources of measurement error: sample, chip to chip, chip re-use, and pixel digitization error. In this version of the simulator - we will NOT model chip re-use variability. We treat each of the other as additive normal error with mean 0 and standard deviation = σ.

Although we could just as easily treat 3 additive errors as one - for conceptual transparency we keep them separate.

Let Sample to Sample Variability error = $\varepsilon_s \sim N(0, \sigma_S)$

Let Chip to Chip error = $\varepsilon_c \sim N(0, \sigma_c)$

Let Pixel digitization error = $\varepsilon_{pd} \sim N(0, \sigma_{pd})$

Thus for each sample s taken from a dish d and measured, new vavalues of ε_s , ε_c , and ε_{pd} are drawn, and the average expression level for each gene Vi at time t is

$$\mathbf{M_s[Vi{:}t(d)]} = \text{Vi:t(d)} + \varepsilon_s + \varepsilon_c + \varepsilon_{pd}$$

If we draw 4 samples from each dish, and don't re-use any chips, then our final data table would look as follows:

		time 1				time t		
Dish.	Chip	V1:1	.	VM:1	.	V1:t	.	VM:t
1	1	M_1[V1:1(d=1)]	.			.	.	M_1[VM:t(d=1)]
1	2	M_2[V1:1(d=1)]						M_2[VM:t(d=1)]
1	3	M_3[V1:1(d=1)]						M_3[VM:t(d=1)]
1	4	M_4[V1:1(d=1)]						M_4[VM:t(d=1)]
2	5	M_1[V1:1(d=2)]						. .
.	.	.	.	.	.	.	.	.
D	4D-3	M_1[V1:1(d=D)]						M_1[VM:t(d=D)]
D	4D-2	M_2[V1:1(d=D)]						M_2[VM:t(d=D)]
D	4D-1	M_3[V1:1(d=D)]						M_3[VM:t(d=D)]
D	4D	M_4[V1:1(d=D)]						M_4[VM:t(d=D)]

Figure 17: After Measurement Error

Measurement Error Parameters

The parameters the user must specify for the measurement error model, with defaults in brackets, are:

1. Dish to Dish variability = **sd(DV)** [10]
2. Number of samples per dish = **S** [4]
3. Sample to Sample Variability σ_S, [.025]
4. Chip to Chip Variability σ_c, [.1]
5. Pixel Digitization σ_{pd}, [.025]

8. References

Edwards, R., & Glass L. (2000). Combinatorial explosion in model gene networks., Chaos, V. 10, N. 3, September., pp. 1-14.