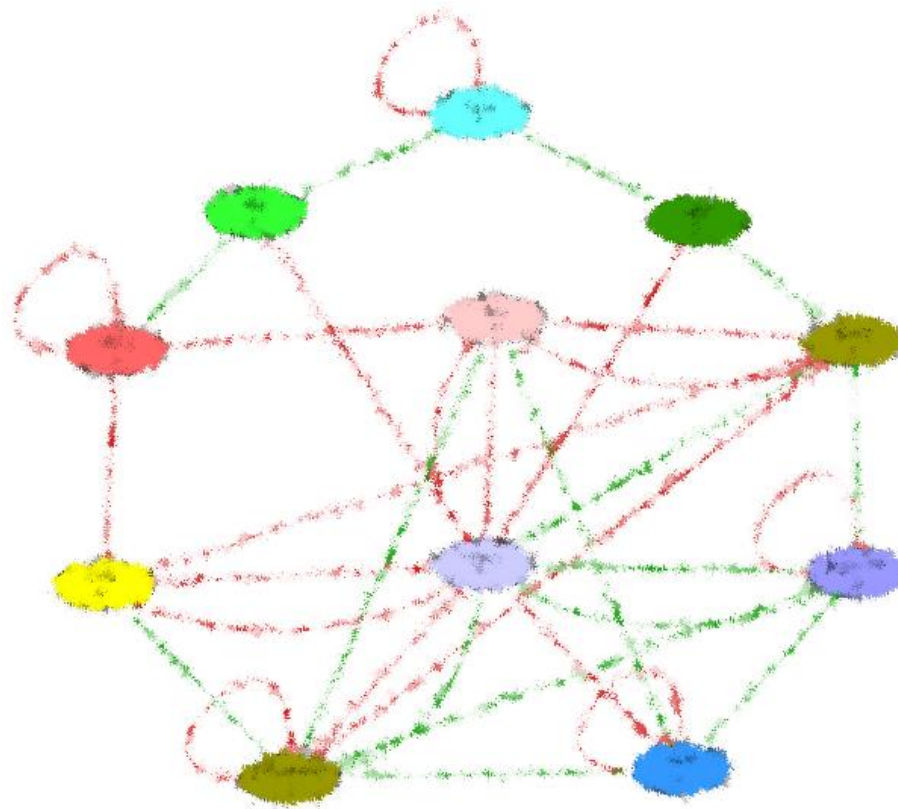


# Graphs and Simulation of Biological Networks

Amir Rubinstein, CS @ TAU



Python Programming for Biologists

# Outline

- Part I – Crash intro to **Graph Theory**
- Part II - **Boolean** model for **regulatory network** simulation

➔ Both topics, as well as others, are taught more elaborately in:

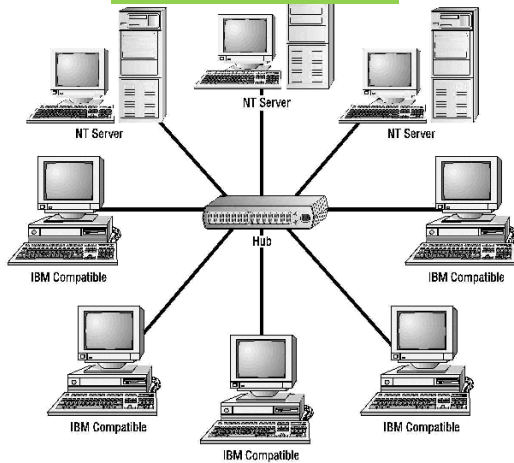
## **Computational Approaches for Life Scientists**

url: [ca4ls.wikidot.com](http://ca4ls.wikidot.com)

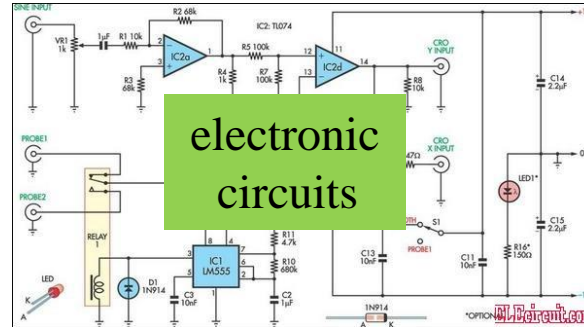
A course designed to enrich Biologists with basic **ideas** and **notions** from Computer Science, **beyond programming and tools**.

# Networks

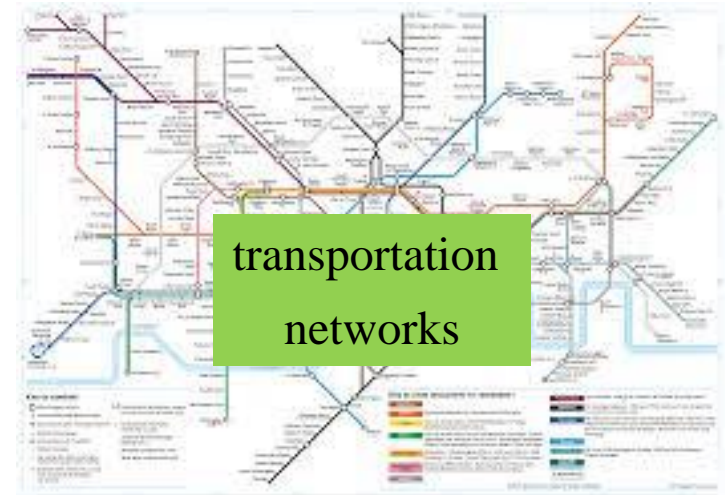
computer  
networks



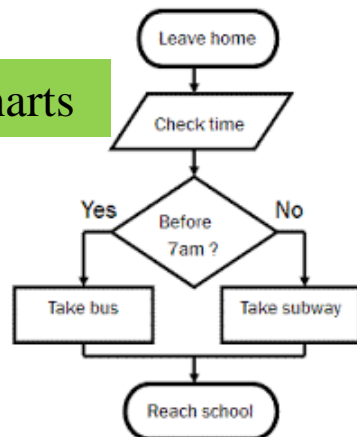
electronic  
circuits



transportation  
networks

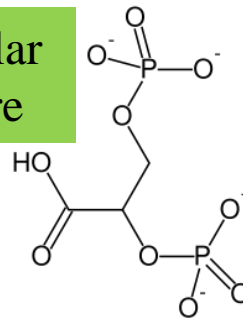


flow charts



and...  
biological  
networks!

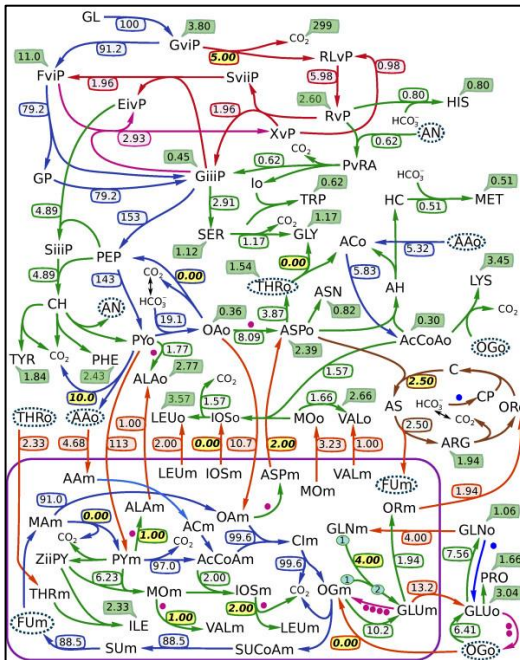
molecular  
structure



social  
networks

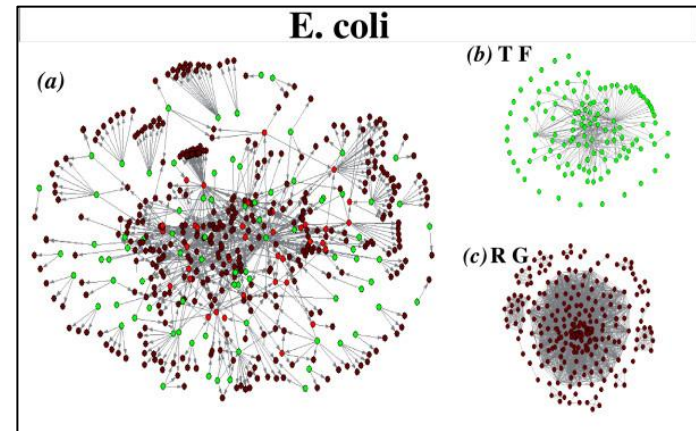


# Biological Networks - Examples



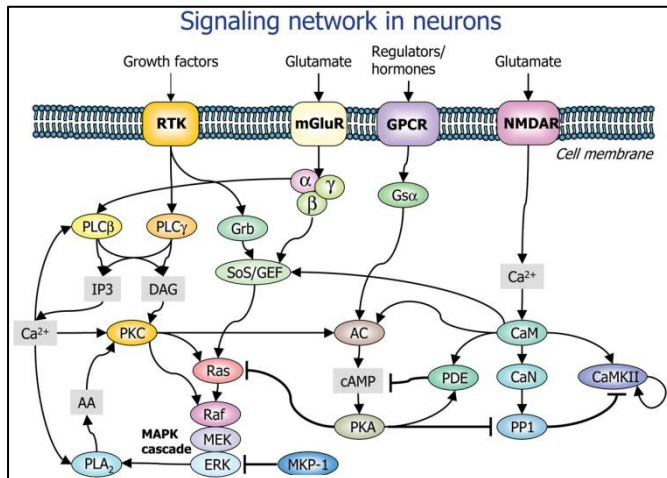
**Metabolic and amino acid biosynthesis pathways of yeast**

[Schryer et al., BMC Systems Biology, \(2011\)](#)



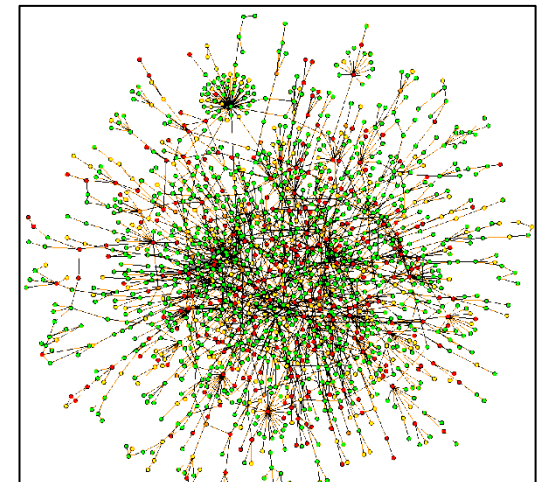
**E. coli transcriptional regulatory network**

[Guzmán-Vargas et al., BMC Systems Biology \(2008\)](#)



**Signaling network in neurons**

[Klipp et al., BMC neuroscience \(2006\).](#)



**The PPI Network in yeast**

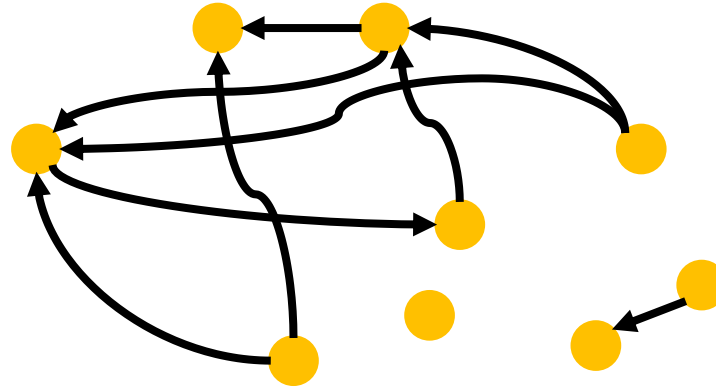
[Jeong et al., Nature \(2001\)](#)

# Visualization and Analysis of Biological Networks

- There are various **tools** and **software packages** for the visualization of networks.
- When the network is large and dense, it is sometimes difficult to extract meaningful information from its visual representation.
- Computational analyses of networks enable valuable insights into their structure, properties and behavior.
- The mathematical structure used to model networks is called a **graph**.  
**Graph theory** deals with studying various aspects of graphs, and we will now introduce the basics of this field.

# Introduction to Graph Theory

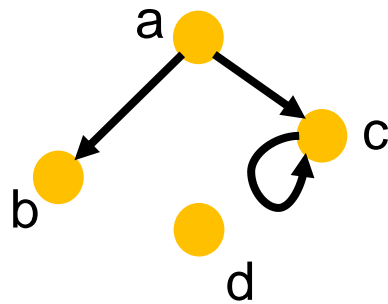
- A **graph** is a set of interactions, or relationships, between pairs of objects.



- The objects are called **nodes**<sup>\*</sup>, and the interactions are termed **edges**<sup>\*\*</sup>.
- If the edges have **directions**, the graph is called a **directed graph** (or **digraph**). Otherwise it is an **undirected** graph.

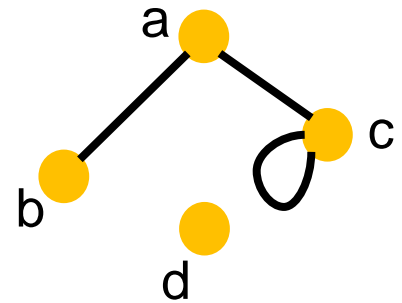
# Graphs – More Formally

- A graph  $G$  is a pair  $G = (V, E)$  where:
  - $V$  is a set of element (called nodes)
  - $E$  is a set of **pairs** from  $V$  (called edges)



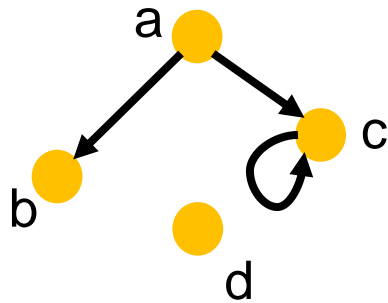
$$V = \{a, b, c, d\}$$

$$E = \{(a, b), (a, c), (c, c)\}$$



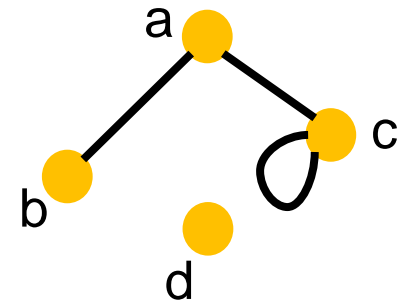
- In an undirected graph we **ignore the order** of nodes in an edge.
- Note: this is a **mathematical** definition.  
It has nothing to do with Python's sets.

# Basic Notions



$$V = \{a, b, c, d\}$$

$$E = \{(a, b), (a, c), (c, c)\}$$

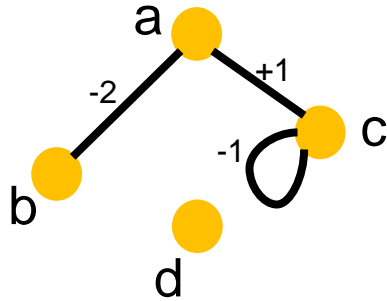


- Common notations:  $|V| = n$  ,  $|E| = m$
- neighboring / adjacent / connected nodes
- neighborhood of a node
- degree of a node (for directed graphs: **in**-degree and **out**-degree)
- endpoints of an edge (for directed graphs: source/target)
- a loop



# Weighted Graphs

- A **weighted graph** is a graph in which edges are assigned **values**, called weights.



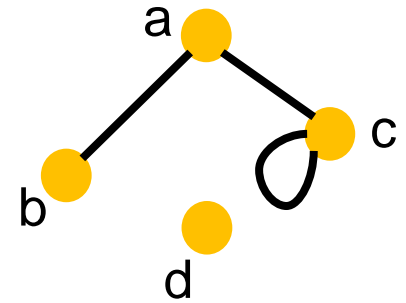
- What can weights resemble in a biological context?

# Paths and Connectivity

- A **path**  $p$  in a graph  $G = (V, E)$  is a sequence of nodes

$$p = (v_1, v_2, \dots, v_k)$$

Such that  $(v_i, v_{i+1}) \in E$  for every  $1 \leq i < k$



If there is a path from  $a$  to  $b$  we say that  $b$  is **reachable** from  $a$ .

If  $v_1 = v_k$  then the path is called a **cycle**.

- The **length** or **weight of a path**  $p$  is the number of edges in it, denoted  $|p|$ .

In **weighted** graphs, this is the **sum of weights** along the path.

- How would you define the **distance** between 2 nodes in a graph?

$$d(v_i, v_j) = ???$$

- A graph is **connected** if there is a path from every node to every other node (in other words every node is reachable from any other node)

# Special Graphs

- Tree

An undirected graph that is:

- connected
- acyclic (= contains no cycles)



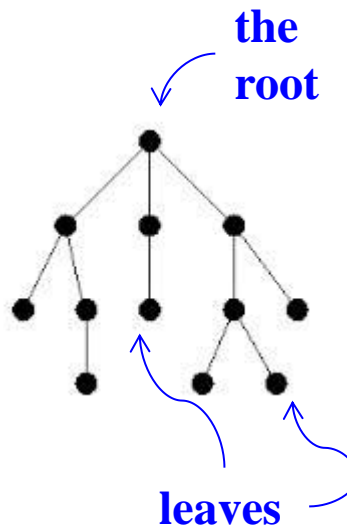
- Rooted tree

A tree with a special node called **root**.

This defines a hierarchy:

- parents and ancestors
- Children and descendants

A **leaf** is a node with no children.



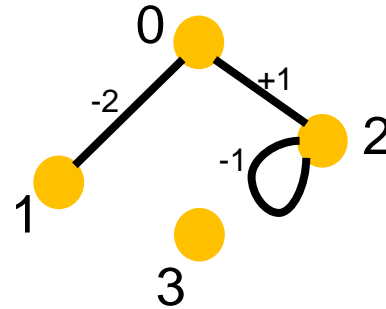
In CS, rooted trees grow downwards...

*(A full binary tree with 16 leaves.  
Courtesy of Dr. Shlomit Pinter,  
photo taken in Kenya, 2005)*

# Graph Representation

- One simple way to represent a graph is a **matrix of adjacencies** (there are additional ways that we will not discuss).

$$G = \begin{bmatrix} [0, -2, 1, 0], \\ [-2, 0, 0, 0], \\ [1, 0, -1, 0], \\ [0, 0, 0, 0] \end{bmatrix}$$



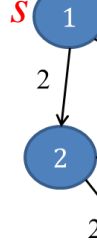
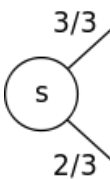
- For example,  $G[1][0] == -2$ , and  $G[3][1] == 0$ .
- If a matrix represents an **undirected** graph, then it must be **symmetric**.

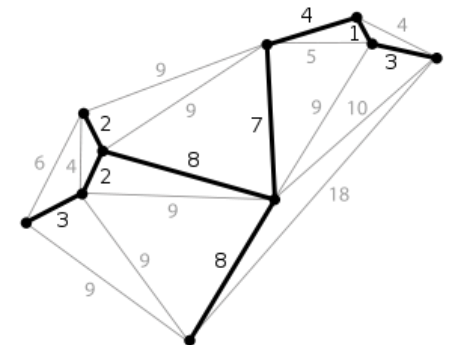
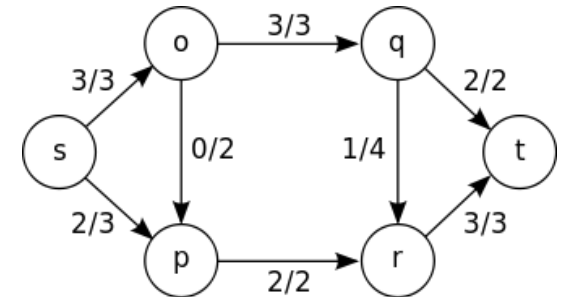
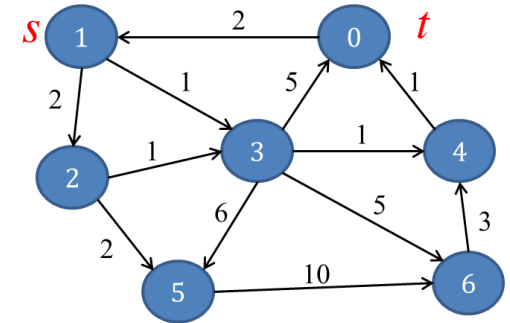
# The Bigger Picture

- Graph theory and graph algorithms are very central within CS.

Computational biologists use graph theory to study properties of biological networks, and graph algorithms to solve biological problems (some examples next).

# Common Problems in Graph Theory

- The **shortest path** problem: find a path from  $s$  to  $t$ , whose “cost” is minimal.
- 
- The **maximal flow** problem: find a maximum feasible flow from  $s$  to  $t$ . (weights are flow capacities).
- 
- The **spanning tree** problem: find a subgraph that is a tree and connects all the vertices, with minimal total weight.
  - You may also want to check out these two famous topics, related to graph theory: the [7 bridges of Königsberg](#), and the [4 color theorem](#).



# Outline

- Part I – crash intro to Graph Theory
- Part II - Boolean model for regulatory network simulation

# Mathematical Models and Simulation

- **Simulation**: an imitation of how a real-world process or system operates over time



A **computer simulation** (aka *in silico* experiments) is a simulation run on computers.

- In biology, computer simulation is used to replace / complement some **tedious** and **costly** lab experiments. It enables conducting numerous “experiments” under various conditions, in a scale that is **infeasible experimentally**.



# Mathematical Models and Simulation (cont.)

- Running a computer simulation requires constructing a **mathematical model** - some formal representation of the system using, e.g., **equations** and **algorithms**.

- A model is an **abstraction** of reality.

A valuable model should capture the relevant aspects of the system, with the appropriate level of detail.

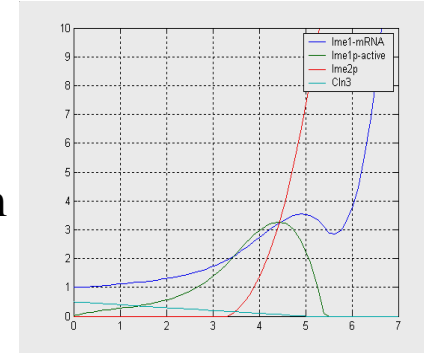
- Advantages of using math. models and simulation in biology:
  - 1) **Descriptive**: forces clarity of expression and precision in describing systems / processes / hypotheses
  - 2) **Analytic**: promotes understanding and provides insights
  - 3) **Predictive**: enables predictions regarding the behavior of the system under various conditions

# Quantitative vs. Qualitative Models

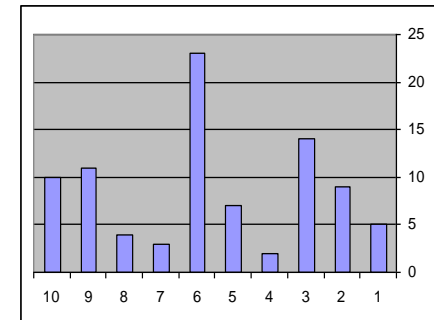
- **Qualitative**\* models predict trends, types of dynamics, and general properties, such as:
  - Robustness to mutations
  - Stability against perturbations (e.g., change in a cell's conditions)
  - Fail-safety
  - Conditions for cyclic behavior
- **Quantitative**\*\* models predict specific values that can be compared to actual experimental data, such as:
  - Kinetics
  - Concentrations
  - Expression levels

# Continuous vs. Discrete Models

- **Continuous**<sup>\*</sup>: infinitely divisible (*e.g.* **reals**)
  - Usually in the form of differential equations
  - Provide a high resolution description of the biological system



- **Discrete**<sup>\*\*</sup>: made of distinct, indivisible units (*e.g.* **integers**)
  - Discrete **time**: time progresses in discrete steps (clock ticks)
  - Discrete **space**: biological quantities are discrete
- Discrete models tend to be simpler,  
more computationally efficiency,  
and require less detailed biological data.

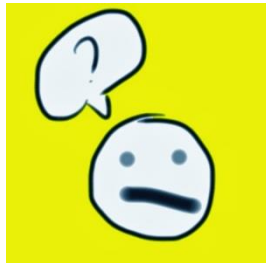


\* Heb: רציף

\*\* Heb: בדיד

# Continuous vs. Discrete Models (cont.)

- What about **biological quantities**? Are they discrete or continuous?  
*e.g.:* interactions, concentrations, reaction times, signals, etc.



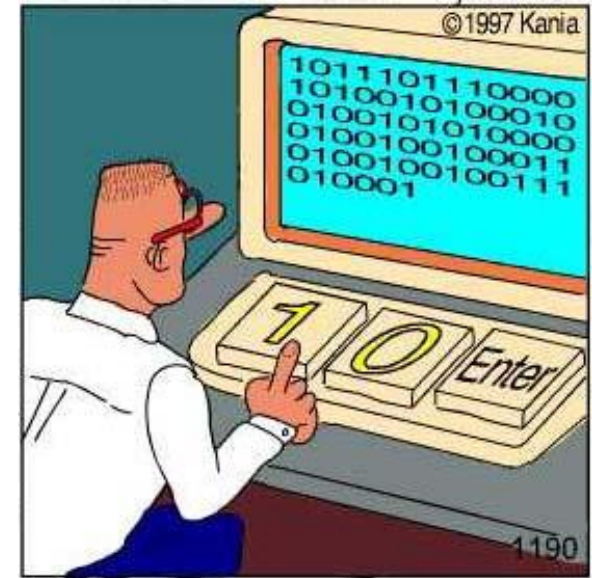
- Whether these are really continuous or discrete is a physical question, or maybe even a philosophical one.  
But anyway, **modeling does not have to conform with the nature of the modeled entity.**

# Case Study – The Yeast Cell-Cycle Boolean Model

- Boolean - 0/1
- A simple case of a discrete model
- Qualitative

Based on the paper:

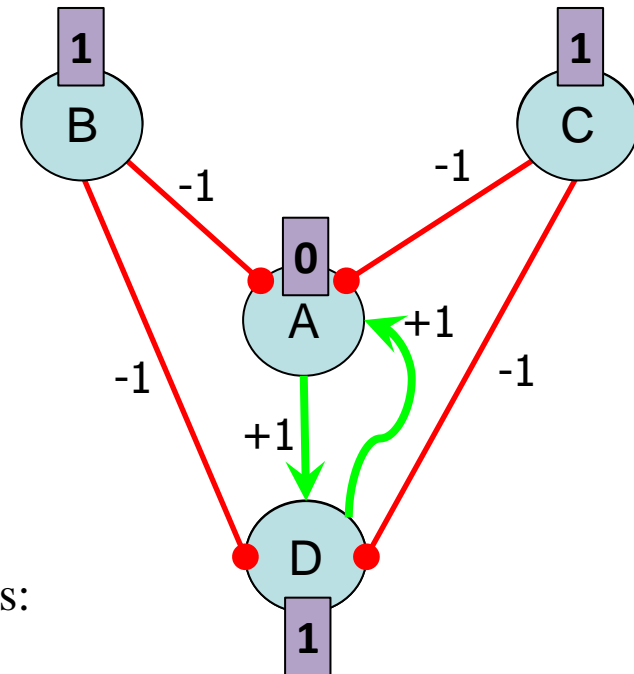
[The yeast cell-cycle network is robustly designed](#),  
*Li et. al., PNAS 2004*



Real programmers code in binary.

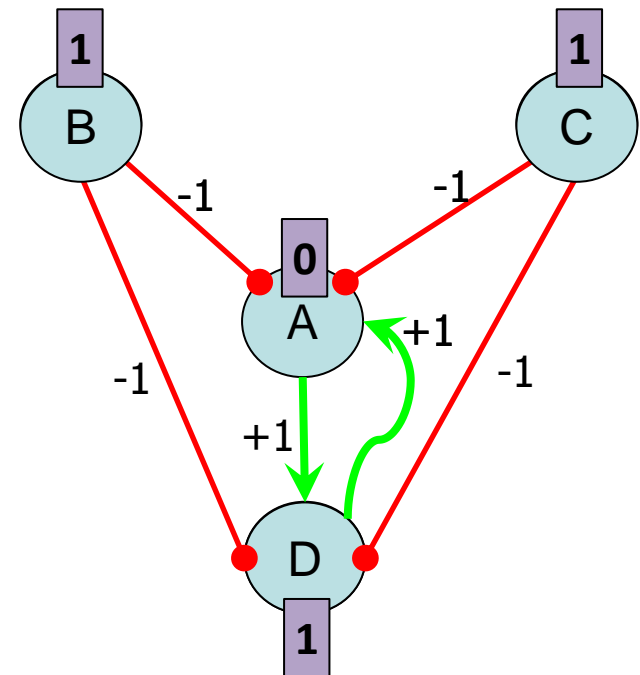
# The Boolean Model – User Input

- The model consists of a graph with states.
- Nodes: can represent proteins, mRNA, nutrients, cellular events (*e.g.* mitosis), external signals (*e.g.* light, injected hormone)
  - Can assume **state 0** (non active) or **1** (active)
  - A **vector** of the network is a sequence of all nodes' states: [0,1,1,1]
  - Each node is given an **initial state**. So the network has an **initial vector**.
- Edges: regulation effects
  - weighted**
  - (+ **activation**)
  - (- **inhibition**)



# The Boolean Model - Simulation

- Time is discrete (time steps = 1,2,3,...)
- A transition function determines the states of nodes in the next time step in a synchronous fashion.  
It moves the system into the next vector.
- Transition function is applied repeatedly, until one of two options:
  - ☐ Steady state (aka "fixed point")  
2 consecutive identical vectors
  - ☐ Infinite loop  
2 **non**-consecutive identical vectors



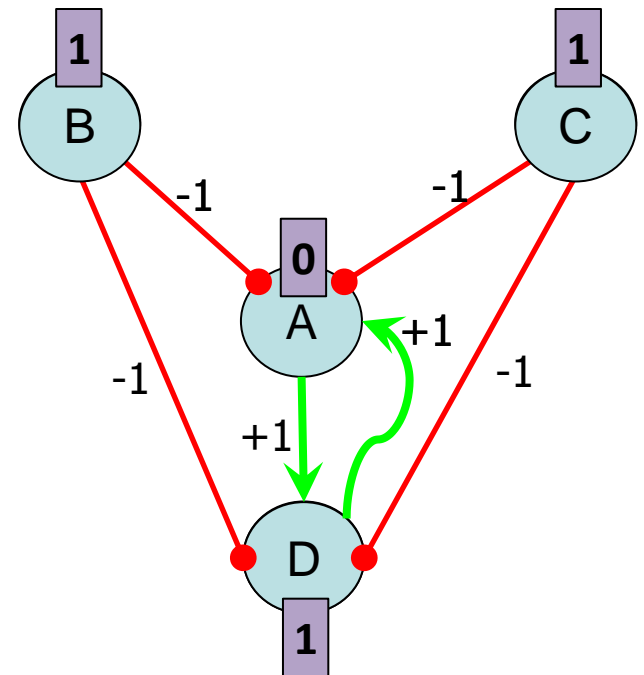
# The Boolean Model – Transition Function

- Transition function:

Sums the effects on each node, caused by all its incoming edges.

$$\sigma_i(t) = \sum_j w(j, i) \cdot s_j(t)$$

weight of edge  $(j, i)$       state of node  $j$  at time step  $t$



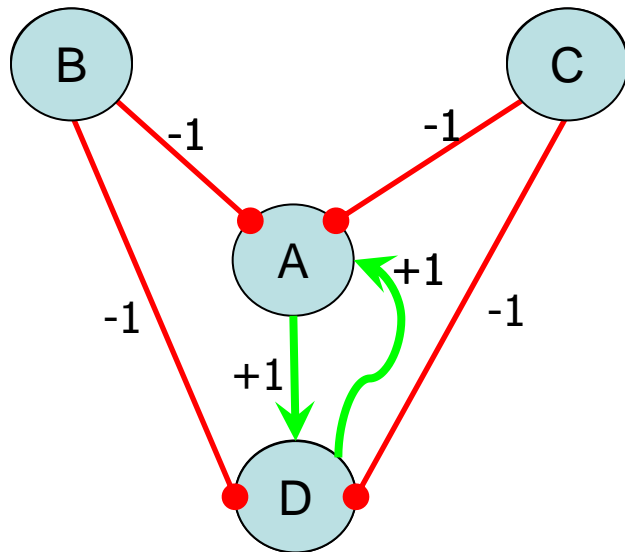
- The state update:

$$s_i(t + 1) = \begin{cases} 1 & \text{if } \sigma_i(t) > 0 \\ 0 & \text{if } \sigma_i(t) < 0 \\ s_i(t) & \text{else} \end{cases}$$



# Example 1

- Let's see what happens to node A at  $t_2$ :



	A	B	C	D
$t_1$ :	1	1	0	0

+

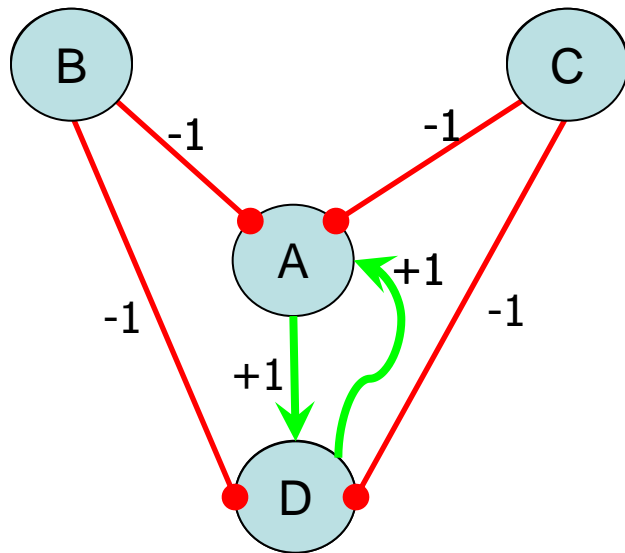
+

= -1

	A	B	C	D
$t_2$ :	0			

# Example 2

- Let's see what happens to node A at  $t_2$ :



	A	B	C	D
$t_1$ :	1	0	0	1

+

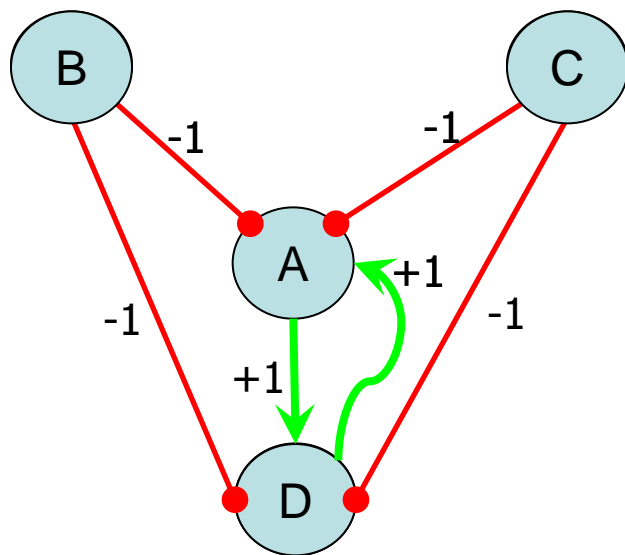
+

= +1

$t_2$ :	1			
---------	---	--	--	--

# Example 3

- Let's see what happens to node A at  $t_2$ :



	A	B	C	D
$t_1$ :	1	0	1	1

+

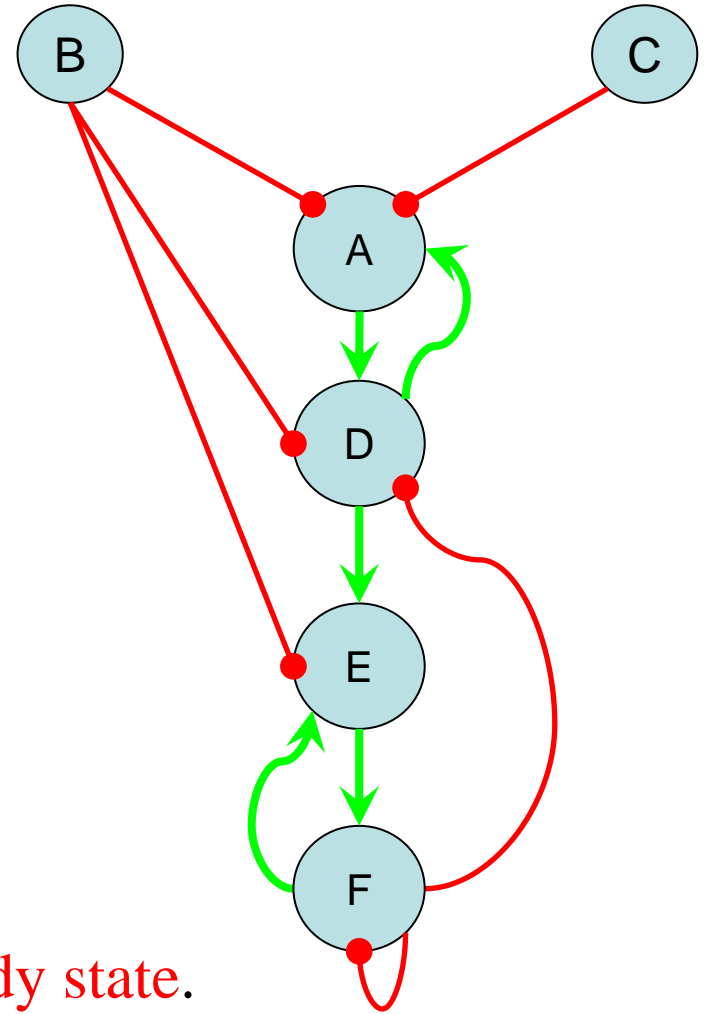
+

= 0

	A	B	C	D
$t_2$ :	1			

# A Simulation - Full Example

	A	B	C	D	E	F
$t_1$ :	1	0	1	0	0	0
$t_2$ :	0	0	1	1	0	0
$t_3$ :	0	0	1	1	1	0
$t_4$ :	0	0	1	1	1	1
$t_5$ :	0	0	1	0	1	1
$t_6$ :	0	0	1	0	1	1



Vectors in  $t_5$  and  $t_6$  are identical → **steady state**.

Can the system get out of this steady-state in the future?

# Exercise - Loops

Give an example for a network and initial vector that yield an **infinite loop**.

Hint: 2 nodes are enough.

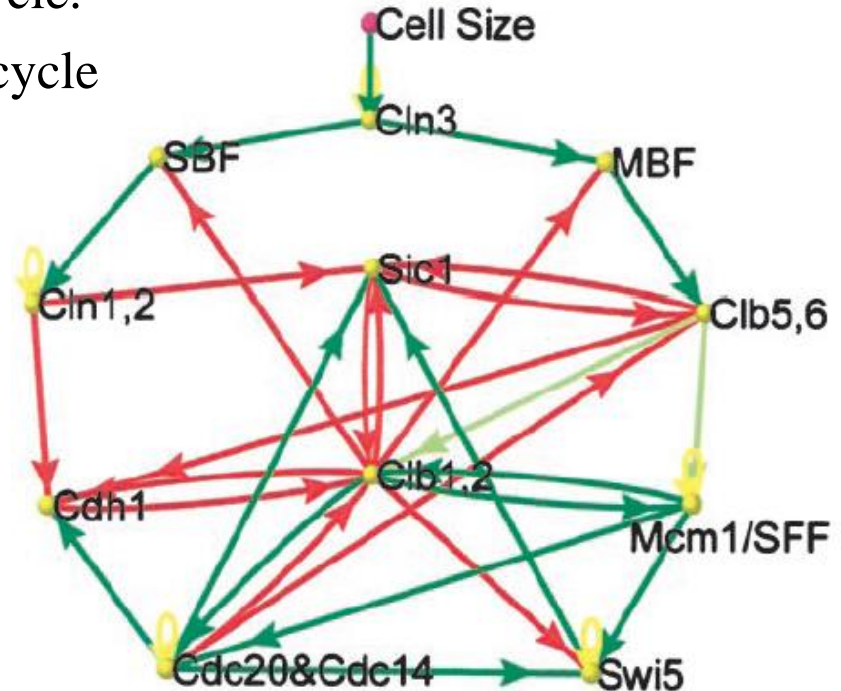
# Simulation of Cell-Cycle in Yeast

- 11 nodes – main regulators of yeast cell-cycle.
- "Cell Size" is the signal for entry into cell-cycle

- Each node can be either 0/1.

Red/yellow edges: weight = -1

Green edges: weight = +1



- Simulation is executed on **all possible initial vectors**.

# How many?

How many potential **fixed points**?

# Yeast Cell-Cycle Simulation **Fixed Points**

- No initial vector yields a loop.
- Out of  $2^{11} = 2048$  potential steady states, only 7 are reached !

Table 1. The fixed points of the cell-cycle network

Basin size	Cln3	MBF	SBF	Cln1,2	Cdh1	Swi5	Cdc20	Clb5,6	Sic1	Clb1,2	Mcm1
1,764	0	0	0	0	1	0	0	0	1	0	0
151	0	0	1	1	0	0	0	0	0	0	0
109	0	1	0	0	1	0	0	0	1	0	0
9	0	0	0	0	0	0	0	0	1	0	0
7	0	1	0	0	0	0	0	0	1	0	0
7	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0	0

The **main "attractor"**:  
this steady state attracts  
~86% of initial states.

# A Complete Cell-Cycle Simulation

- Start with a vector representing **stationary G<sub>1</sub>** condition but with Cln3=1 (signal to initiate cell cycle).

Time	Cln3	MBF	SBF	Cln1,2	Cdh1	Swi5	Cdc20 and Cdc14	Clb5,6	Slc1	Clb1,2	Mcm1/SFF	Phase
1	1	0	0	0	1	0	0	0	1	0	0	START
2	0	1	1	0	1	0	0	0	1	0	0	G <sub>1</sub>
3	0	1	1	1	1	0	0	0	1	0	0	G <sub>1</sub>
4	0	1	1	1	0	0	0	0	0	0	0	G <sub>1</sub>
5	0	1	1	1	0	0	0	1	0	0	0	S
6	0	1	1	1	0	0	0	1	0	1	1	G <sub>2</sub>
7	0	0	0	1	0	0	1	1	0	1	1	M
8	0	0	0	0	0	1	1	0	0	1	1	M
9	0	0	0	0	0	1	1	0	1	1	1	M
10	0	0	0	0	0	1	1	0	1	0	1	M
11	0	0	0	0	1	1	1	0	1	0	0	M
12	0	0	0	0	1	1	0	0	1	0	0	G <sub>1</sub>
13	0	0	0	0	1	0	0	0	1	0	0	Stationary G <sub>1</sub>

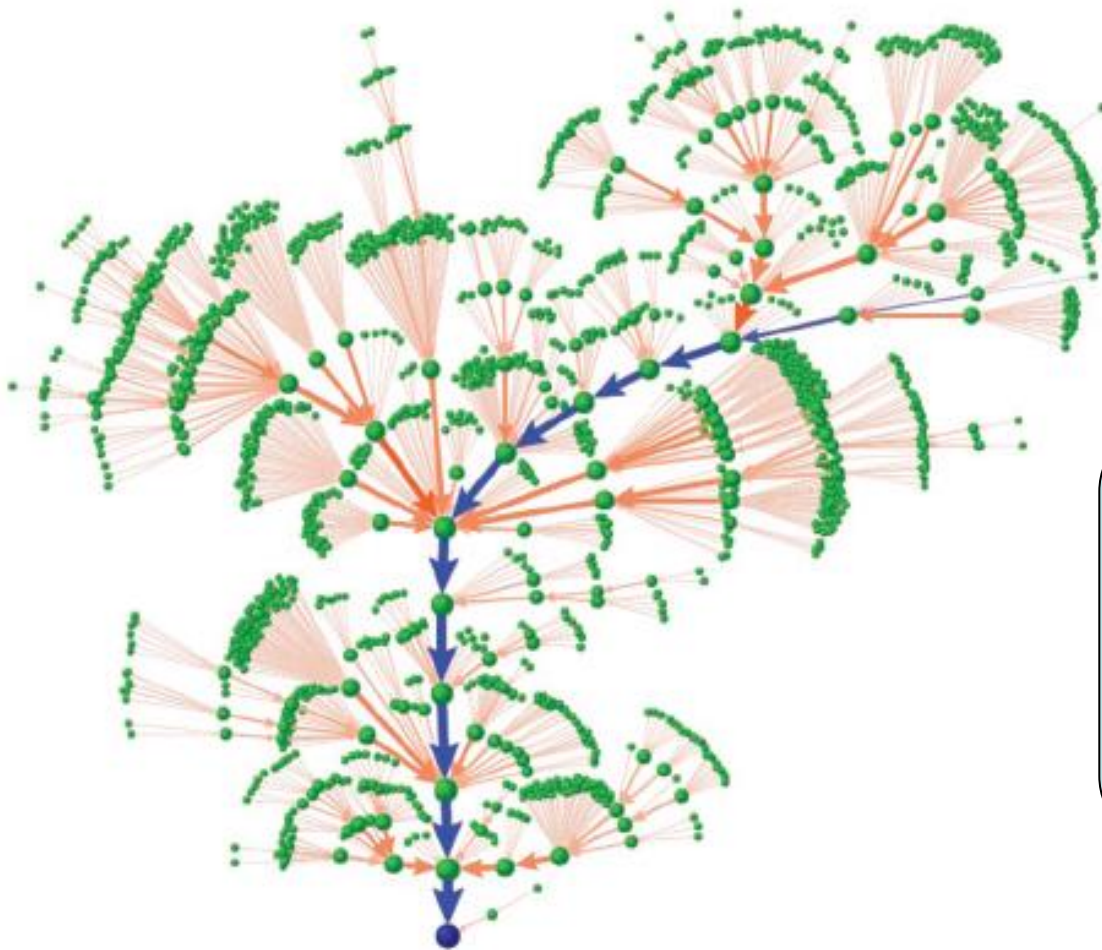
The right column indicates the cell-cycle phases. Note that the number of time steps in each phase do not reflect its actual duration.

- As indicated in the table, this simulation is compatible with the **cell cycle stages**:  $G_1 \rightarrow S \rightarrow G_2 \rightarrow M \rightarrow G_1$  (stationary)



# Cell-Cycle in Yeast - Transitions Tree

- Each **node** in the tree represents a **vector**, **edges** represent **transitions** in the simulation.



Transition tree for the  
main "attractor"

1764 = 86% of initial vectors.  
(there are 6 other, smaller trees)

From *The yeast cell-cycle network is robustly designed*, Li et. al., PNAS 2004.

Tree drawn with Pajek software (<http://vlado.fmf.uni-lj.si/pub/networks/pajek>)

# Cell-Cycle in Yeast - Paper **Conclusions**

- The yeast cell-cycle is **stable**.

Computational observation: with high probability, **changes to the initial vectors** yield the same fixed point.

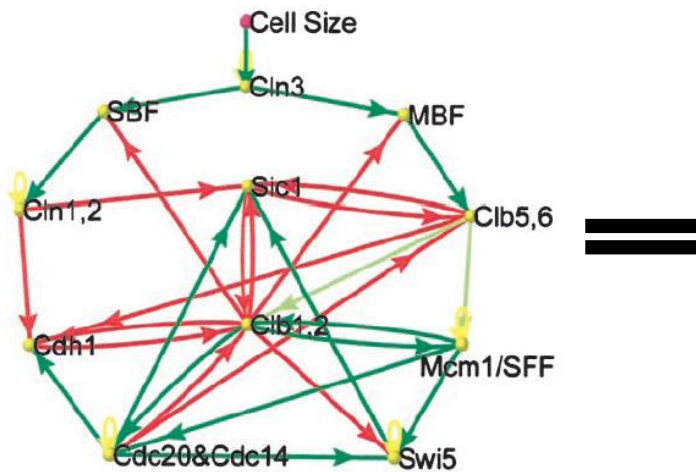
- The yeast cell-cycle is **robust**.

Computational observation: with high probability, **small changes in the network structure** (insert/delete node, change edge) will not harm cell cycle behavior.

# Implementation of the Boolean Model in Python

- We will now introduce / write some of the code that implements the Boolean model.
- We will not deal with detecting loops.

# Representation of the Yeast CC Network



```
nodes =
['Cln3', 'MBF', 'SBF', 'Cln1,2', 'Cdh1', 'Swi5',
'Cdc20/14', 'Clb5,6', 'Sic1', 'Clb1,2', 'Mcm1/SFF']
```

```
G = [
[-1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[ 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
[ 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
[ 0, 0, 0, -1, -1, 0, 0, 0, -1, 0, 0, 0],
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0],
[ 0, 0, 0, 0, 0, -1, 0, 0, 0, 1, 0, 0],
[ 0, 0, 0, 0, 1, 1, -1, -1, 1, -1, 0, 0],
[ 0, 0, 0, 0, -1, 0, 0, 0, -1, 1, 1, 0],
[ 0, 0, 0, 0, 0, 0, 0, -1, 0, -1, 0, 0],
[ 0, -1, -1, 0, -1, -1, 1, 0, -1, 0, 1, 0],
[ 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, -1]]
```

# Implementation Design

- We will divide the task into three functions:

```
def run(G, init):  
    ''' Start with init vector, and run until steady state  
        Does not detect loops for the moment  
        Return the fixed point vector '''
```

*calls*

```
def update(G, states):  
    ''' Return the next states vector -  
        apply transition function to all nodes '''
```

*calls*

```
def update_node(G, states, i):  
    ''' Return new state of node i given states vector '''
```

# Tracking the Execution

- Function should normally avoid printing, unless printouts are carefully controlled.
- We will add this option now.

Additional parameter,  
set by default to False

```
def run(G, init, track = False):  
    ''' Start with init vector, and run until steady state  
        Does not detect loops for the moment  
        Return the fixed point vector  
        Track = True for tracking the execution '''
```

- Should users want to track the execution, they will call:

```
>>> run(G, init, track=True)
```

# Simulation - Run!

```
nodes = ['Clb3', 'MBF', 'SBF', ... ]

init = [1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0]

G = [
    [-1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
    [ 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
    [ 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
    [ 0, 0, 0, -1, -1, 0, 0, 0, -1, 0, 0],
    [ 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0],
    [ 0, 0, 0, 0, 0, -1, 0, 0, 1, 0, 0],
    [ 0, 0, 0, 0, 1, 1, -1, -1, 1, -1, 0],
    [ 0, 0, 0, 0, -1, 0, 0, 0, -1, 1, 1],
    [ 0, 0, 0, 0, 0, 0, 0, -1, 0, -1, 0],
    [ 0, -1, -1, 0, -1, -1, 1, 0, -1, 0, 1],
    [ 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, -1]
]

print("A single simulation of the Yeast cell cycle:")
print(nodes)
run(G, init, track=True)
```

# Simulation - Output

A single simulation of the Yeast cell cycle:

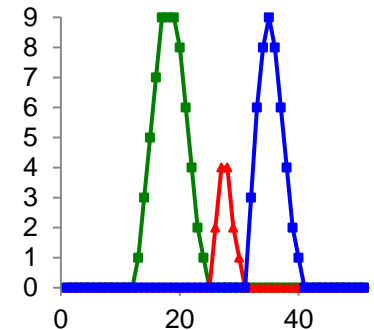
```
['Cln3', 'MBF', 'SBF', 'Cln1,2', 'Cdh1', 'Swi5', 'Cdc20/Cdc14', 'Clb5,6',  
'Sic1', 'Clb1,2', 'Mcm1/SFF']
```

```
[1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0]  
[0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0]  
[0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0]  
[0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]  
[0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0]  
[0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1]  
[0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1]  
[0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1]  
[0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1]  
[0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1]  
[0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0]  
[0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0]  
[0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0]
```



# Model Extension 1: Discrete State Space

- Instead of 0/1, nodes can now assume states between  $0, \dots, U$  (e.g.  $U=9$ )
- $U=1$  is the special case of the Boolean model we saw



- Transition function changes accordingly:

$$s_i(t+1) = \begin{cases} \min(U, s_i(t) + 1) & \text{if } \sigma_i(t) > 0 \\ \max(0, s_i(t) - 1) & \text{if } \sigma_i(t) < 0 \\ s_i(t) & \text{else} \end{cases}$$

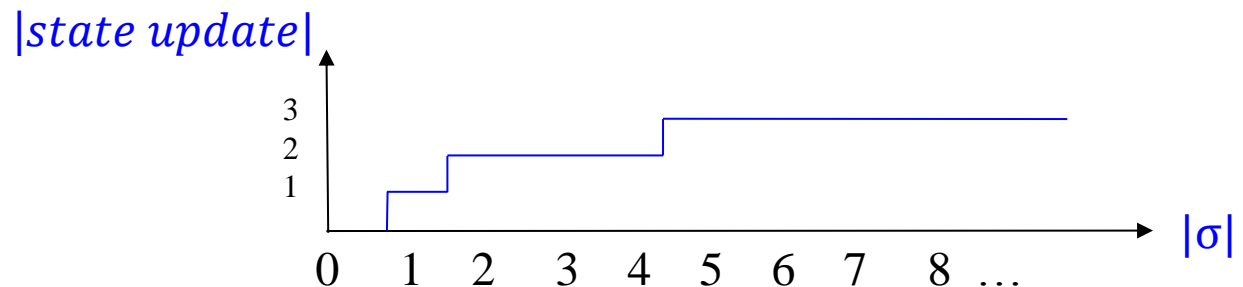
$$\left( \sigma_i(t) = \sum_j w(j, i) \cdot s_j(t) \right)$$

# Model Extension 2: State **Update Function**

- States change by  $\pm 1$  no matter what.
- We may prefer the change to be proportional to  $\sigma_i(t)$ .

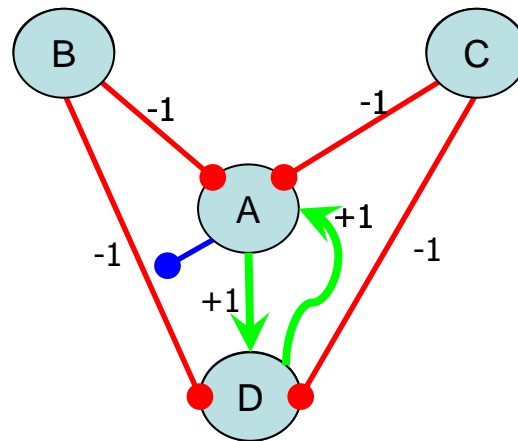
$$s_i(t + 1) = \begin{cases} \min(U, s_i(t) + ?) & \text{if } \sigma_i(t) > 0 \\ \max(0, s_i(t) - ?) & \text{if } \sigma_i(t) < 0 \\ s_i(t) & \text{else} \end{cases}$$

- One reasonable option is a **logarithmic**-order update:



# Model Extension 3,4,...

- A new type of interactions called **dependency** edges: **nodes may block or enable edges**



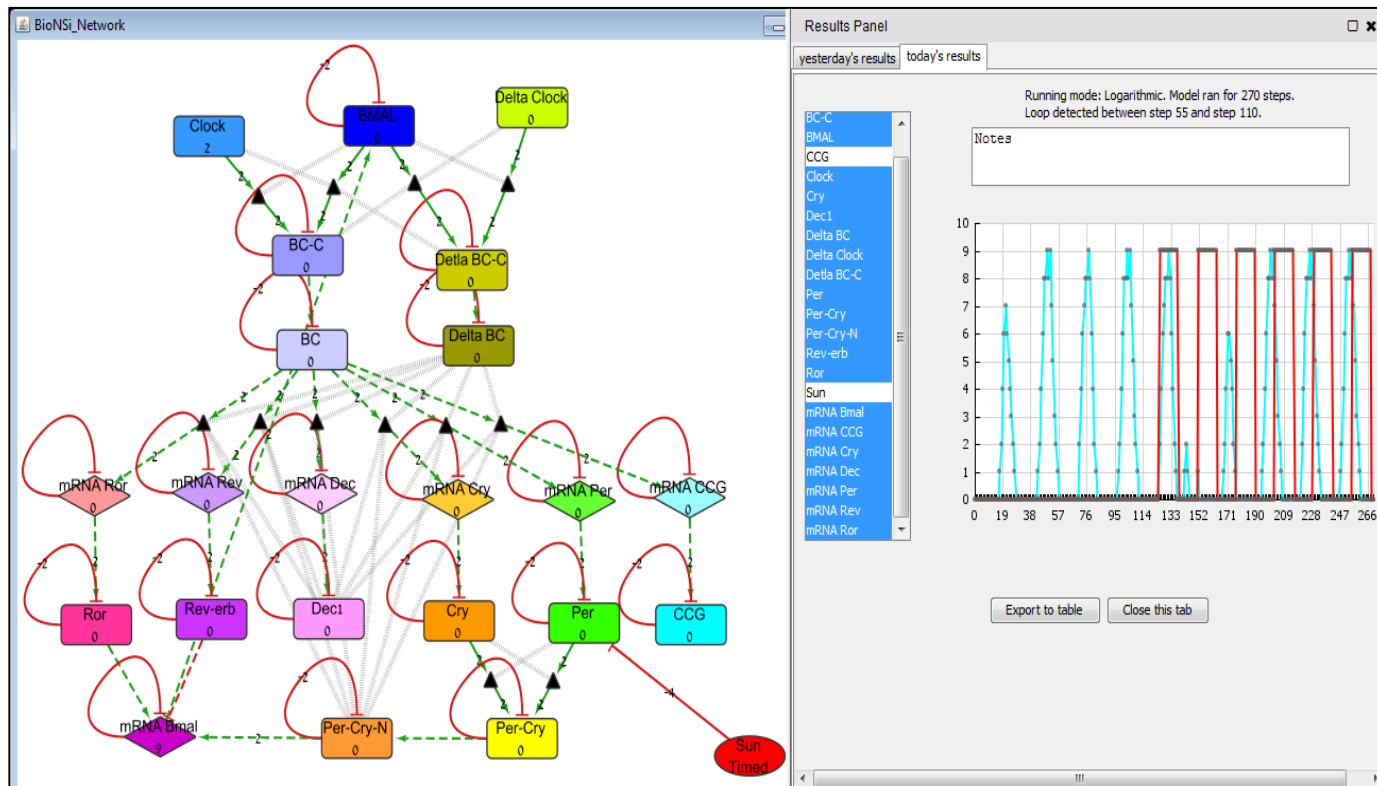
If  $A > 0$  edge  $B \rightarrow D$  is blocked

- **Delays** on edges
  - The effect of node a on node b will occur in later steps

***פרסומת אחת וחזרנו...***

# *BioNSi* – Biological Network Simulator

- A plugin (app) of the popular software **Cytoscape** ([www.cytoscape.org](http://www.cytoscape.org))  
Download from: <http://bionsi.wix.com/bionsi>
- Extends the Boolean model in several ways, including those mentioned



The circadian clock in mammals in a day-night regime, in *BioNSi*

# Reflection: Constructing the “Right” Model?

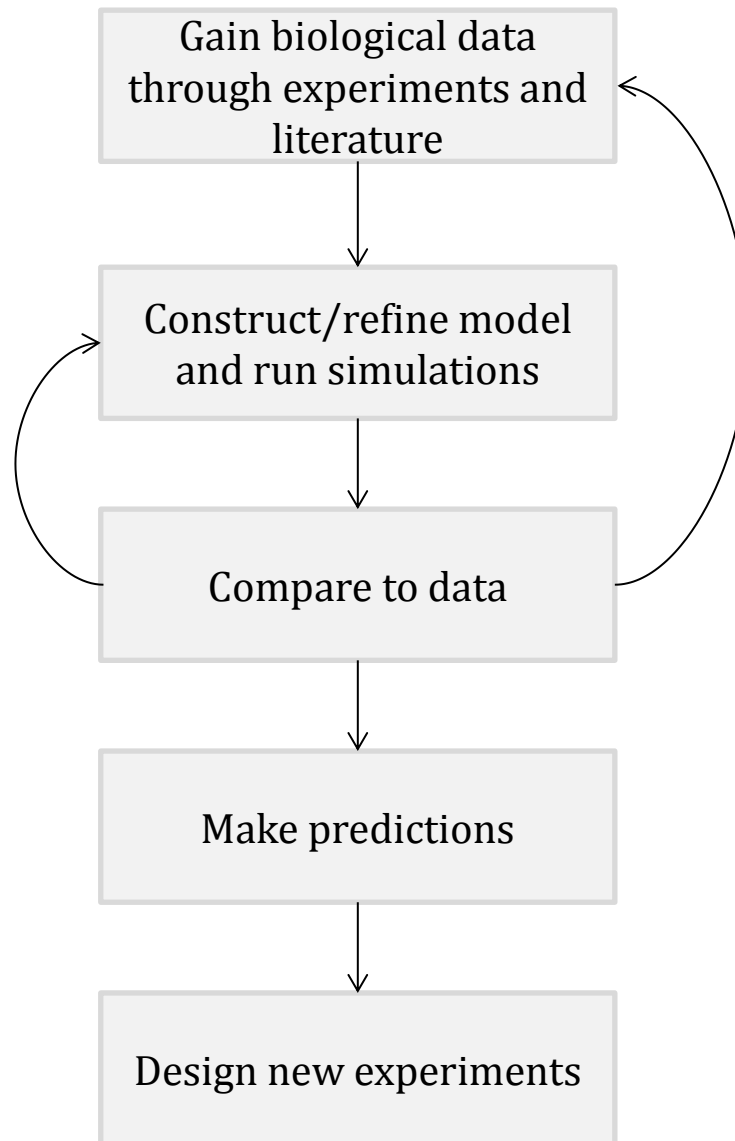


- Main considerations in constructing mathematical models in biology:

Consideration	Meaning	Beware of
Modeling technique	Which model type is best suited for the data and goals of study?	Intractable approaches, too simplified, no sufficient data, etc.
Scope	What elements should be included in the model?	Excluding essential elements or including irrelevant ones
Detail	What level of detail should the model contain?	Too fine- or course-grained models
Parameters	How to set the model parameters appropriately?	Parameter overfitting

- A model often can be used to indicate **gaps in our biological understanding**: a model that fails to recapitulate known biological data reveals where our understanding needs improvement.
- Recall that even “incorrect” models may make correct predictions

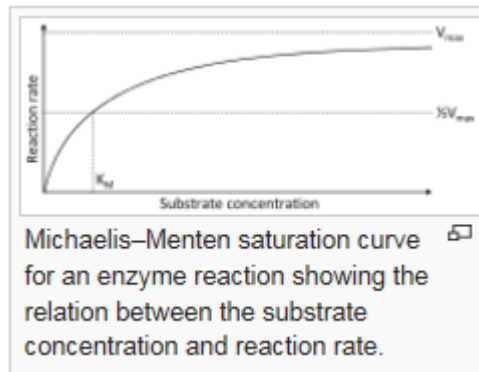
# Reflection: Iterative Simulation-Experiment Approach



# Reflection: Discrete Models



- One major choice in this course's **design** was to exclusively concentrate on **discrete** notions (such as graphs, strings, digital images, FSM, etc.), and less on **continuous** notions.
- Discrete notions are highly underrepresented in **life science curricula**, where **continuous** notions (such as equations over the reals) and probability are taught more widely.



$$v = \frac{d[P]}{dt} = \frac{V_{\max}[S]}{K_M + [S]}$$



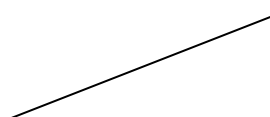
# Some References on Modeling in Biology

- These references provide high level reviews on modeling and simulation in biology:
  - **A book on the interface between biology and computing:** Lin, Herbert S., and John C. Wooley, eds. Catalyzing inquiry at the interface of computing and biology. National Academies Press, 2005. Chapter 5. Link: <http://www.nap.edu/read/11480/chapter/7#p2000dec59970117001>
  - **A review on modeling with the example of a transcription factor signaling network:** Kearns, Jeffrey D., and Alexander Hoffmann. "[Integrating computational and biochemical studies to explore mechanisms in NF- \$\kappa\$ B signaling](#)." Journal of Biological Chemistry 284.9 (2009): 5439-5443.
  - **A review of modeling networks in biology:** Karlebach, Guy, and Ron Shamir. "Modelling and analysis of gene regulatory networks." Nature Reviews Molecular Cell Biology 9.10 (2008): 770-780.
  - **A review on combined experimental-computational study:** Morelli, Luis G., et al. "Computational approaches to developmental patterning." Science 336.6078 (2012): 187-191.

# Appendix: Attractor Size

- Python's **itertools** package: creating iterators for efficient looping (<https://docs.python.org/3.4/library/itertools.html#module-itertools>)

```
def attractor_size(G, attractor):  
    ''' count how many initial vectors end up in the attractor '''  
    n = len(G)  
    cnt = 0  
    total = 0  
  
    all_vectors = itertools.product([0,1], repeat=n)  
  
    for states in all_vectors:  
        states = list(states) #convert tuple-->list, to make mutable  
        final = run(G, states)  
        if final == attractor:  
            cnt+=1  
            total+=1  
    return cnt/total*100
```



Cartesian product  $(0,1)^n$

# Simulation - Run!

```
nodes = ['Cln3', 'MBF', 'SBF', ... ]

G = [
  [-1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
  [ 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
  [ 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
  [ 0, 0, 0, -1, -1, 0, 0, 0, -1, 0, 0],
  [ 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0],
  [ 0, 0, 0, 0, 0, -1, 0, 0, 1, 0, 0],
  [ 0, 0, 0, 0, 1, 1, -1, -1, 1, -1, 0],
  [ 0, 0, 0, 0, -1, 0, 0, 0, -1, 1, 1],
  [ 0, 0, 0, 0, 0, 0, 0, -1, 0, -1, 0],
  [ 0, -1, -1, 0, -1, -1, 1, 0, -1, 0, 1],
  [ 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, -1]
]

final = [0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0] #main attractor
print(round(attractor_size(G, final)), "% of initial vectors end at", final)
```

- Output:

86 % of initial vectors end at [0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0]