Boolean Regulation Networks

Amir Rubinstein, CS@TAU

Advisor: Prof. Benny Chor

Python Programming for Biologists
Tel-Aviv University

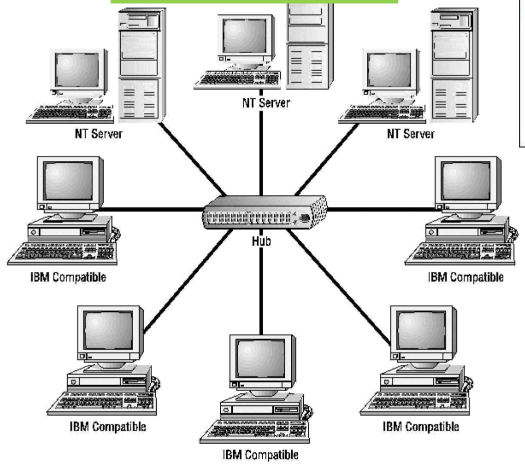
20 May 2015

Outline

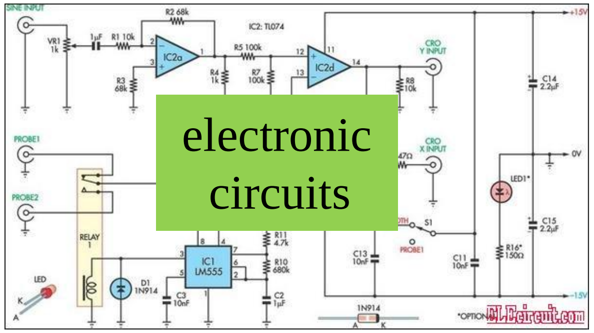
- Part I – Crash intro to Graph Theory
- Part II - Boolean model for regulatory network simulation

Networks

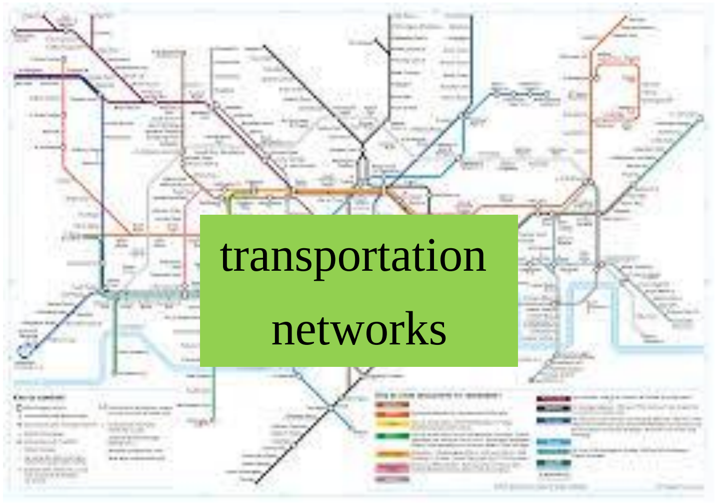
computer networks



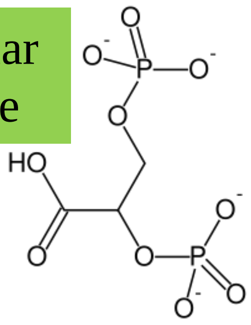
electronic circuits



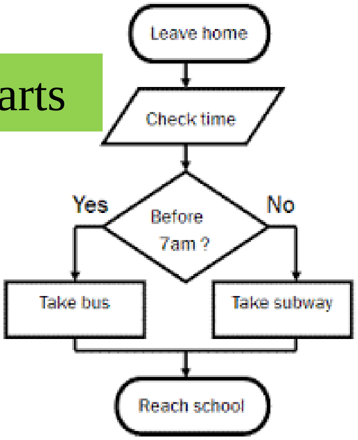
transportation networks



molecular structure



flow charts

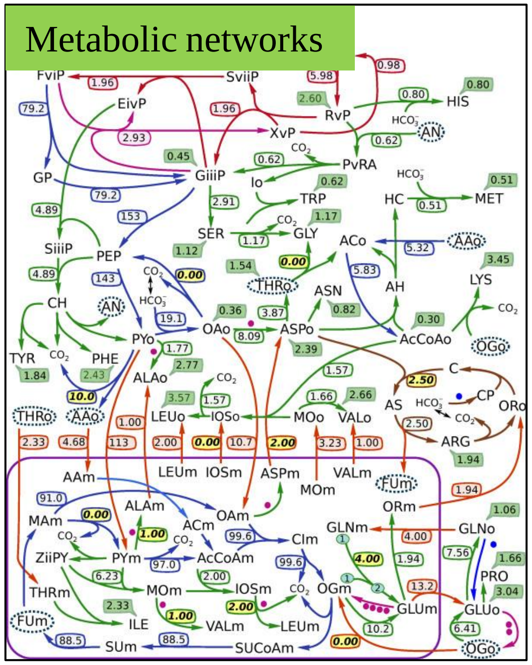


and…

biological networks!

social networks

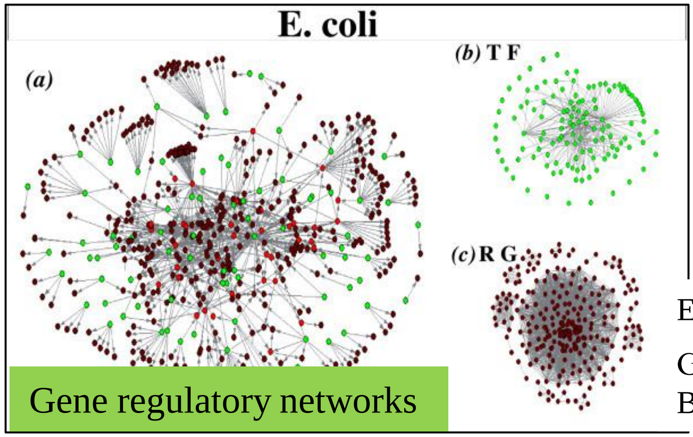
Biological Networks



Metabolic networks

Metabolic and amino acid biosynthesis pathways of yeast

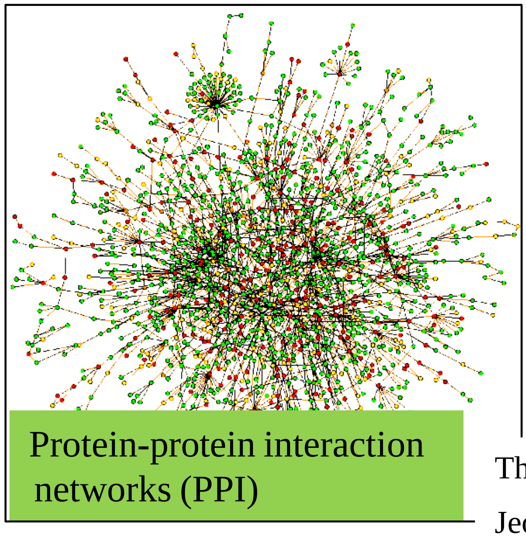
Schryer et al. BMC Systems Biology, no. 1. p. 81 (2011)



Gene regulatory networks

E. coli transcriptional regulatory network

Guzmán-Vargas et al., BMC Systems Biology 2008, 2:13



Protein-protein interaction networks (PPI)

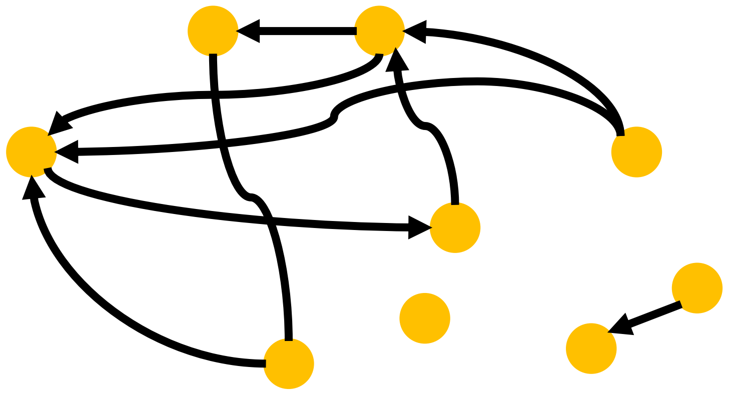
The PPI Network in yeast

Jeong et al. Nature 411, 41 - 42 (2001)

- There are various tools for visualization of networks. But when the networks are large and dense, it is hard to extract information this way.
- This calls for automated methods (algorithms)

Introduction to Graph Theory

- In Computer Science, a graph is a set of interactions, or relationships, between objects.



- The objects are called nodes*, and the interactions are termed edges**.

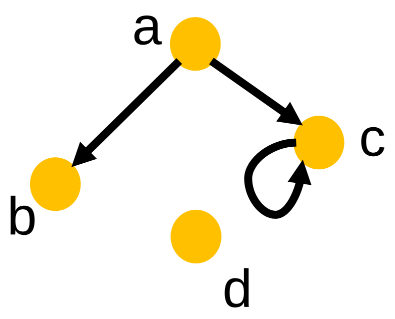
- If the edges have directions, the graph is called a directed graph (or digraph). Otherwise it is an undirected graph.

* or vertices (sg. vertex). Hebrew: קדקד / צומת

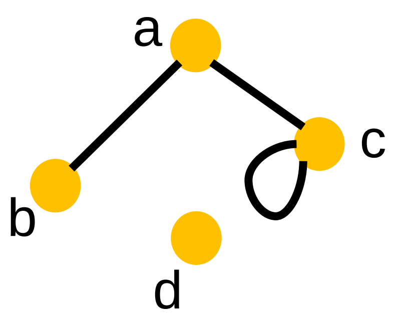
** or arcs, links, chains. Hebrew: צלע / קשת

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)



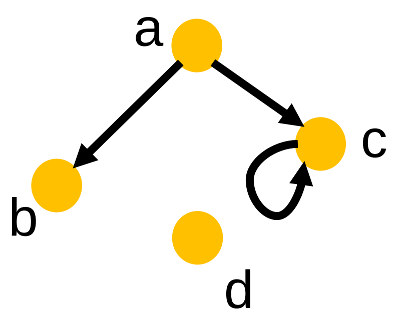
```
V = {a,b,c,d}

E = {(a,b),(a,c),(c,c)}
```



- In an undirected graph we ignore the order of nodes in an edge.

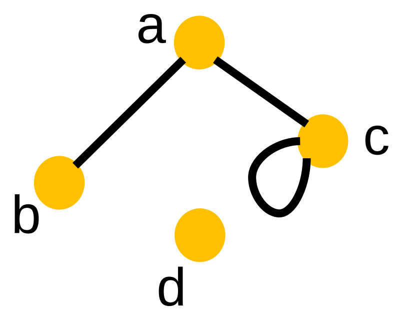
- Note: this is a mathematical definition.
  It has nothing to do with Python's sets.

Node and Edge Notions



`V = {a,b,c,d}`

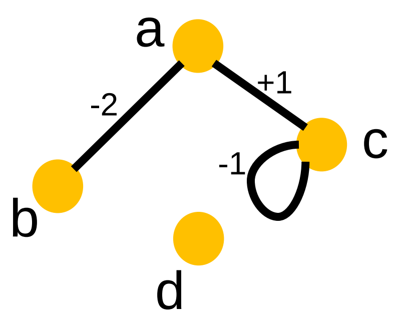
`E = {(a,b),(a,c),(c,c)}`



- `|V| = n , |E| = m`
- neighboring / adjacent /connected nodes
- neighborhood of a node
- degree of a node (digraphs: in-degree and out-degree)
- endpoints of an edge (digraphs: source/target)
- loop

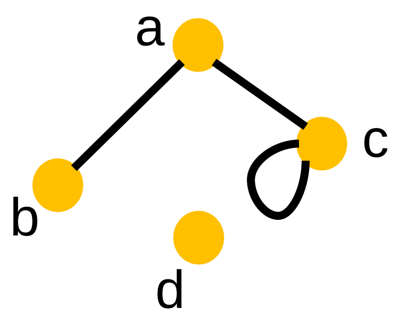
Weighted Graphs

- A weighted graph is a graph in which edges are assigned numeric 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

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

  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 of a path is the number of edges in it.

  Denoted $|p|$.

  In weighted graphs, this is the sum of weights along it (aka weight of a path).

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

  denoted $dist(v_i , v_j)$ or $d(v_i , v_j)$.

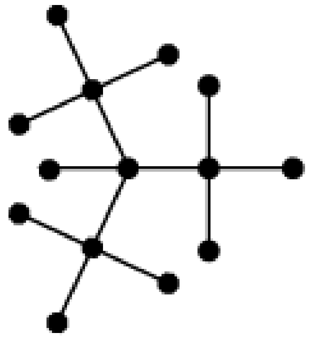
- A graph is connected if there is a path from every node to every other node.

Special Graphs

- Tree

  An undirected graph that is:

  - connected
  - acyclic (= contains no cycles)



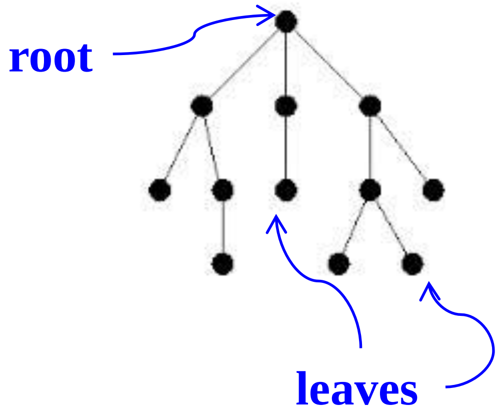
- Rooted tree

  A tree with a special node called the root.

  This defines a hierarchy:

  - parent, ancestor
  - child, descendant

  A leaf is a node with no children.

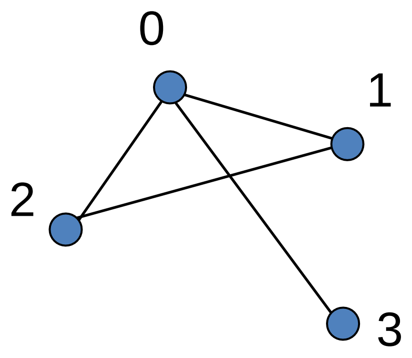


In CS, rooted trees grow downwards…

Graph Representation

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

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



- What can you say about a matrix representing an undirected graph?
- How would you add weights information?

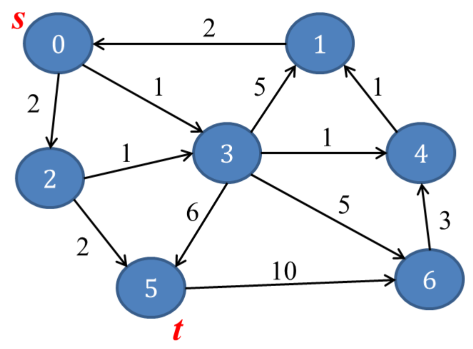
The Bigger Picture

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

  Computational biologists use graph theory to study properties of biological networks (*e.g*. compare them to random ones), and graph algorithms to solve biological problems (some examples next).

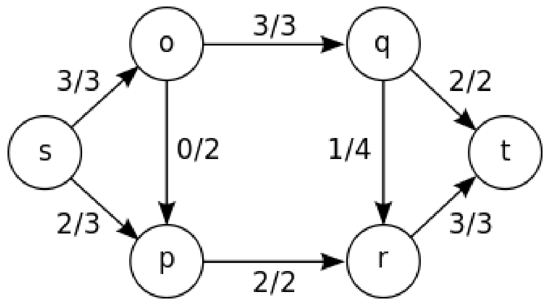
Common Problems in Graph Theory

- The shortest path problem: find a path from *s* to *t* in *G*, whose "cost" is minimal.

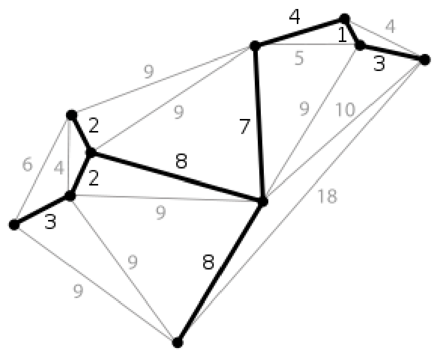


- The maximal flow problem: find a maximum feasible flow from *s* to *t*.

  (weights are flow capacities).



- The spanning tree problem: find a subgraph that is a tree and connects all the vertices, with minimal total weight.



- Many others…

Outline

- Part I – crash intro to Graph Theory

- Part II - Boolean model for regulatory network simulation

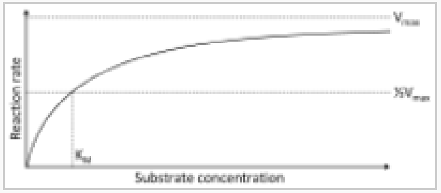
Continuous* vs. Discrete** Models

- A computer model refers to the algorithms and equations used to capture the behavior of the system being modeled.

- Continuous: infinitely divisible (*e.g.* reals)

  Continuous models are usually in the form of defining relations between variables.

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

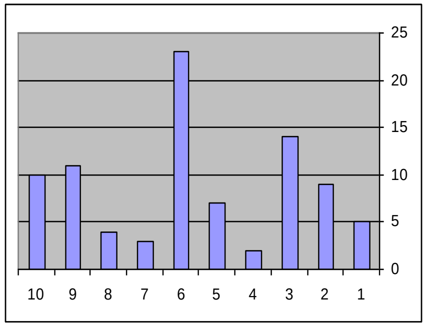


Michaelis–Menten saturation curve for an enzyme reaction showing the relation between the substrate concentration and reaction rate.

- Discrete: made of distinct, indivisible units (*e.g.* integers)

  Manners in which a model can be discrete:

  - Discrete time: time progresses in discrete steps (clock tics)
  - Discrete space: biological quantities are discrete



* He: בדיד ** He: רציף

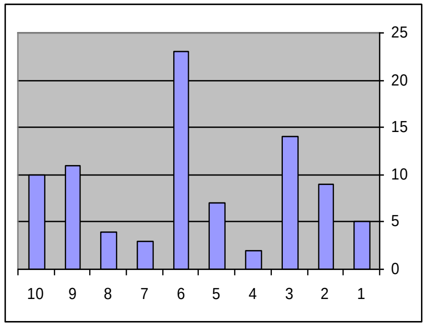
Continuous vs. Discrete Models

- 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.

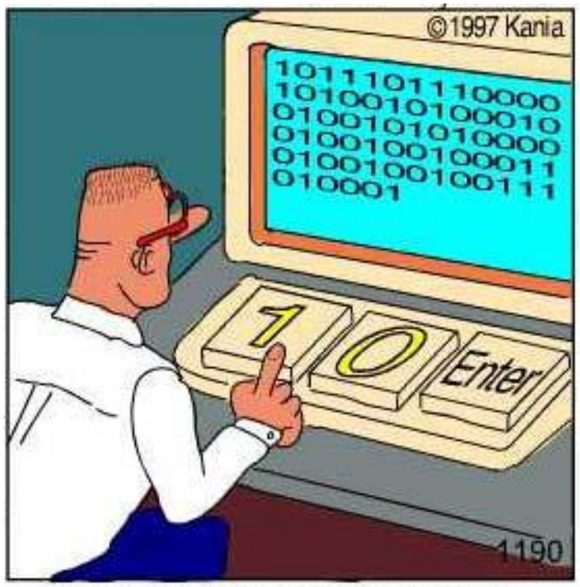


- So why discrete models?
  - Simplicity
  - Computational efficiency
  - Lack of detailed biological data

Boolean models

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

- We will now introduce a Boolean model, used for the simulation of regulatory networks.

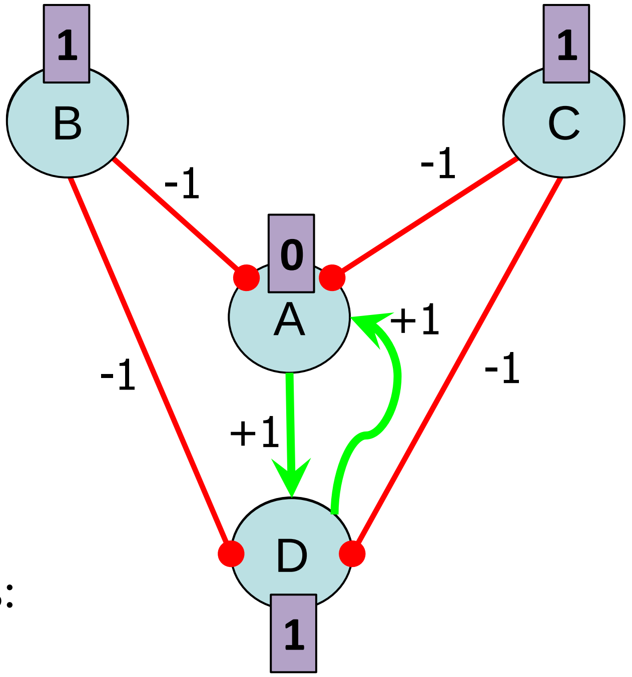


Real programmers code in binary.

Based on the paper:
The yeast cell-cycle network is robustly designed,
Li et. al., PNAS 2004

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

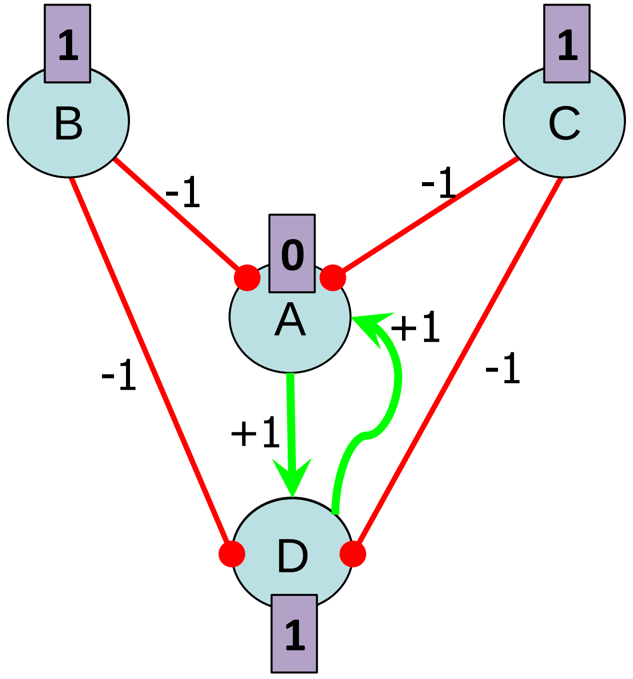
- Time is discrete (time steps = 1,2,3,…)

- 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

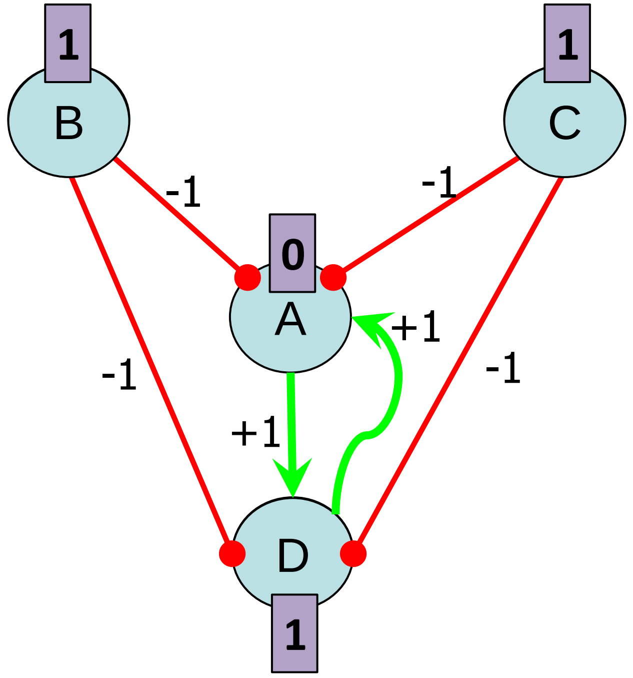


The Boolean Model – Transition Function

- Transition function:

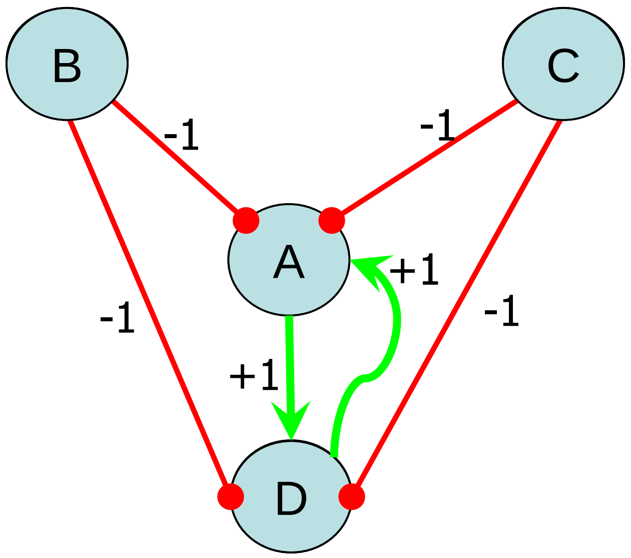
  Sums the effects on each node, caused by all its incoming edges.

```
sum>0 → state=1
sum<0 → state=0
sum=0 → no change
```



Example 1

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



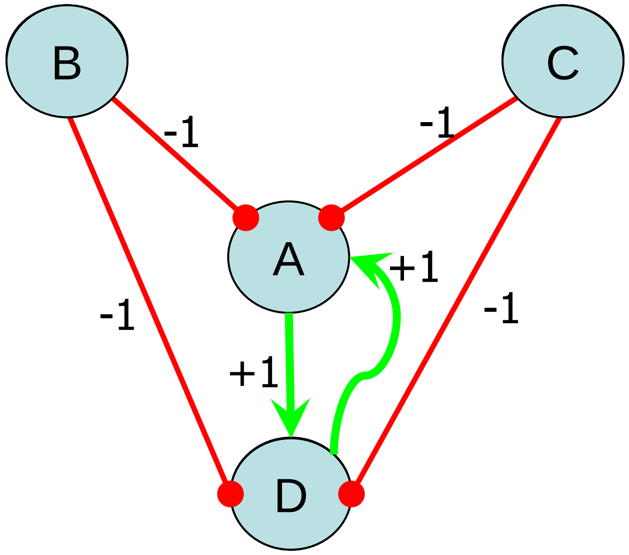
	A	B	C	D
t_1:	1	1	0	0

+ + = -1

t_2:	0			

Example 2

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



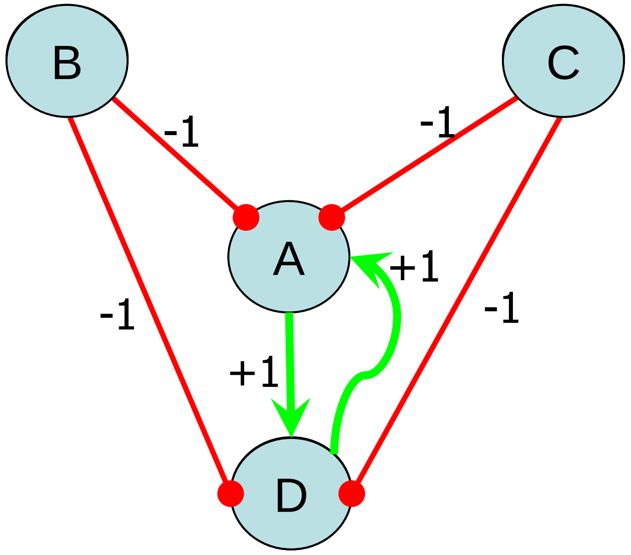
	A	B	C	D
t_1:	1	0	0	1

+ + = +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

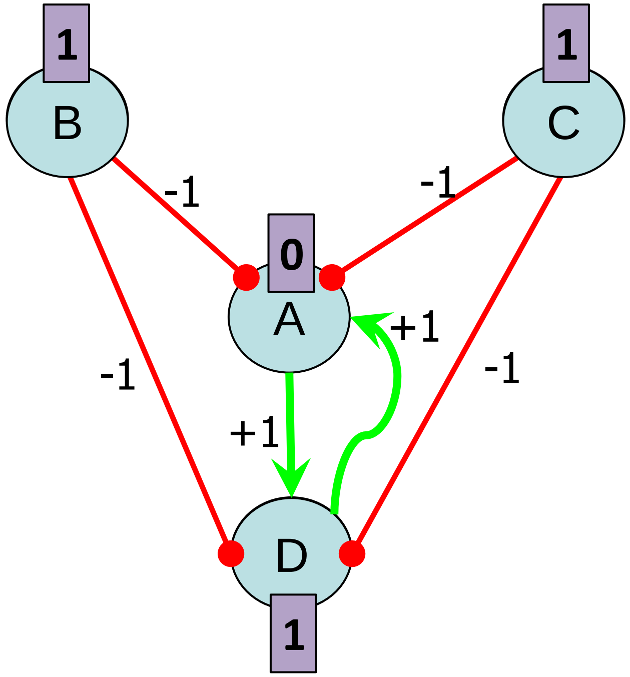
t_2:	1			

The Boolean Model – Transition Function

- Transition function:

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

```
sum>0 → state=1
sum<0 → state=0
sum=0 → no change
```



- In mathematical notation:

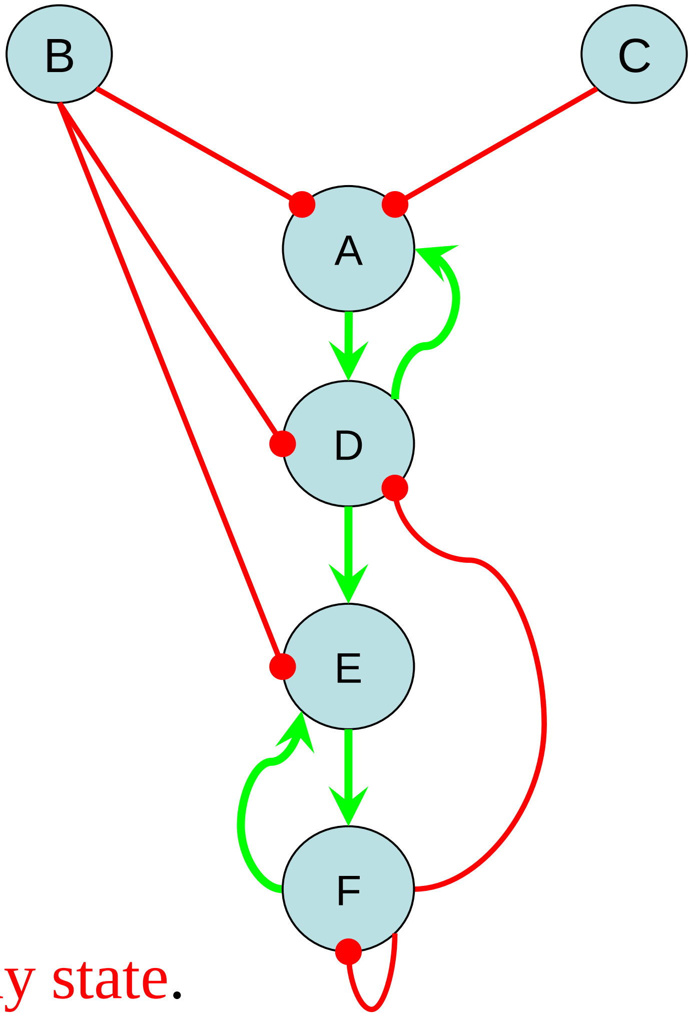
$$s_i(t+1) = \begin{cases} 1 & \sum_j w(j,i)\cdot s_j(t) > 0 \\ 0 & \sum_j w(j,i)\cdot s_j(t) < 0 \\ s_i(t) & else \end{cases}$$

$s_i(t)$ - state of node *i* at time step *t*

$w(j,i)$ - weight of edge (*j* ,*i*)

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?

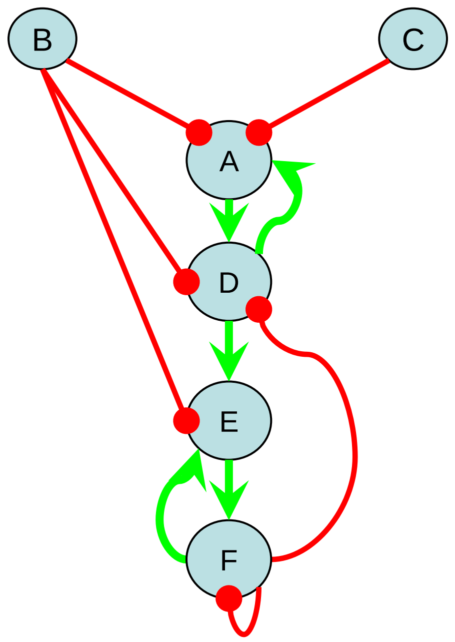
Exercise - Loops

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

Hint: 2 nodes are enough.

Representation of the Data

- The network will be represented as an adjacency matrix, plus a list of nodes names:



=

```
nodes = ['A','B','C','D','E','F']

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

- A vector will be represented as a list:

t_1:	1	0	1	0	0	0
t_2:	0	0	1	1	0	0

.
.
.

=

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

.
.
.

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.

Additional parameter, set by default to False

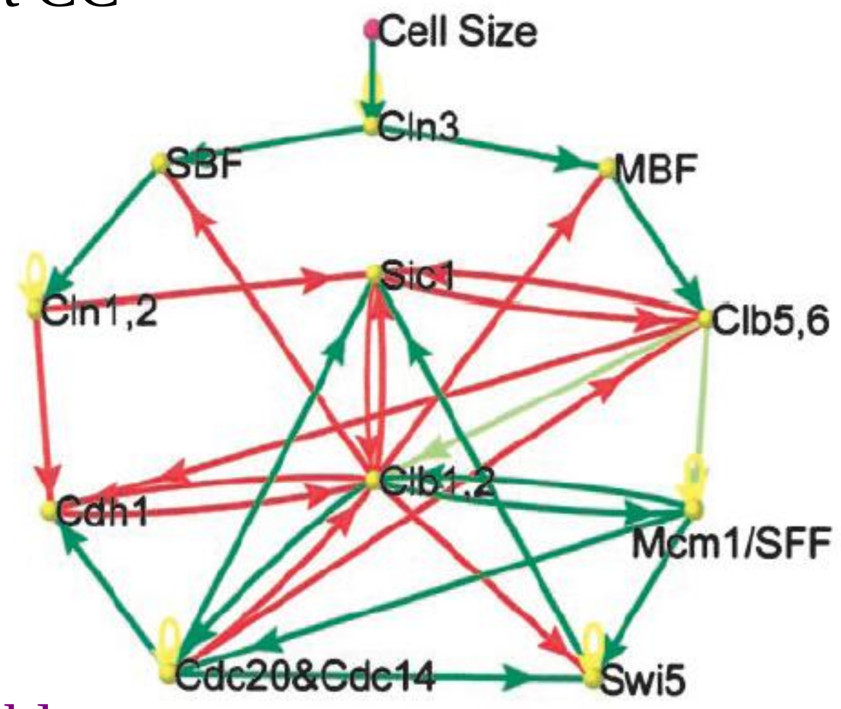
```
def run(G, init, track = False):
    ''' Start with init vector, and run until steady state
        Does not detect loops for the moment
        Return the fixed point vector
        Track = True for tracking the execution '''
```

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

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

Biological Example: Cell-Cycle in Yeast*

- 11 nodes – main regulators of yeast CC
- "Cell Size" is the signal
  for entry into CC
- Red/yellow edges: weight = -1
  Green edges: weight = +1
- Simulation is executed on all possible initial vectors. How many?
  How many potential fixed points?



* Based on the paper: *The yeast cell-cycle network is robustly designed*, *Li et. al., PNAS 2004*

Cell-Cycle in Yeast - Fixed Points

- Out of $2^{11} = 2048$ potential steady states, only 7 are reached !

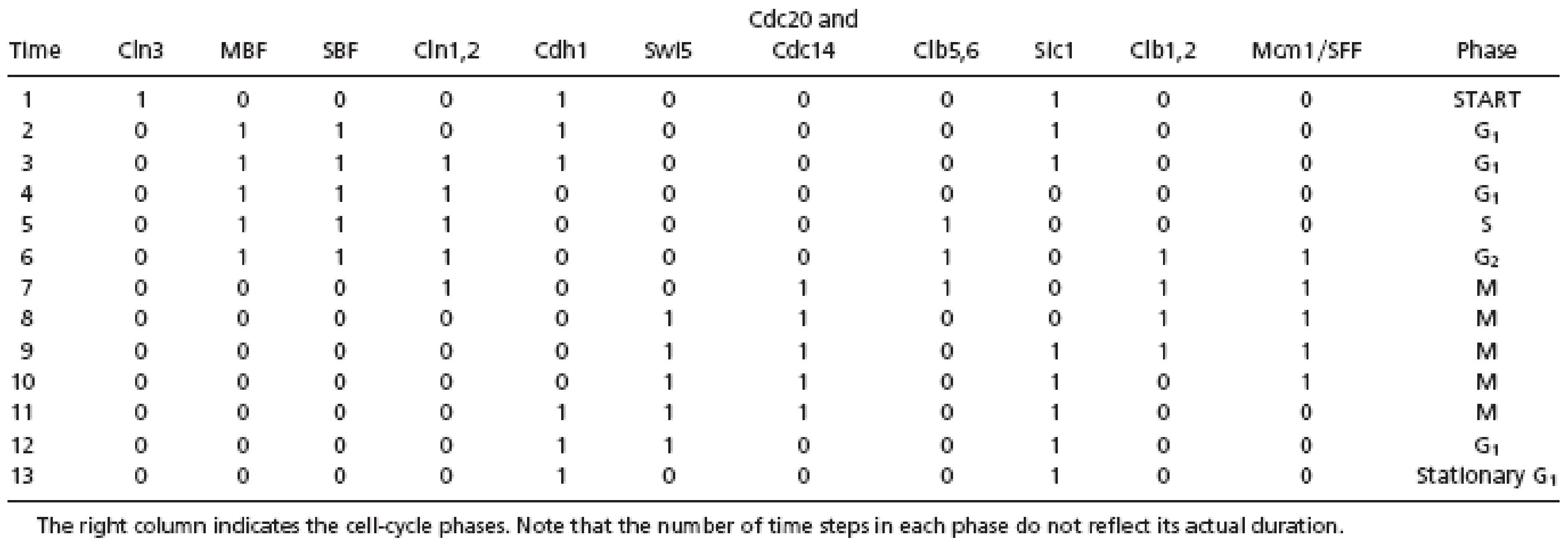
Table 1. The fixed points of the cell-cycle network

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.

Cell-Cycle in Yeast - A Full Simulation

- Representative initial vector that reaches the major attractor:



Time	Cln3	MBF	SBF	Cln1,2	Cdh1	Swi5	Cdc20 and Cdc14	Clb5,6	Sic1	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_1
3	0	1	1	1	1	0	0	0	1	0	0	G_1
4	0	1	1	1	0	0	0	0	0	0	0	G_1
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_2
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_1
13	0	0	0	0	1	0	0	0	1	0	0	Stationary G_1

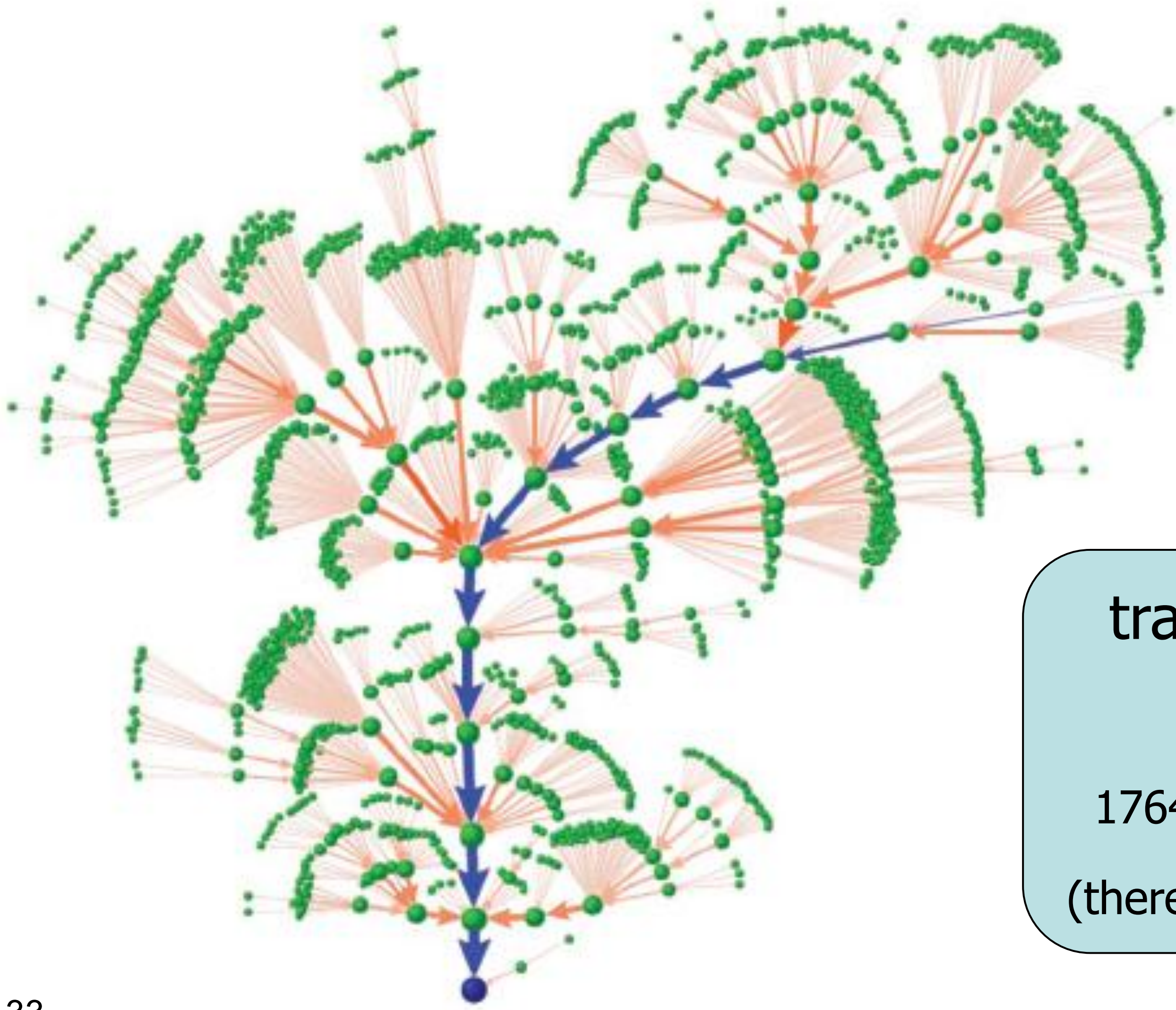
The right column indicates the cell-cycle phases. Note that the number of time steps in each phase do not reflect its actual duration.

- This simulation is compatible with the cell cycle stages:

$$G_1 \rightarrow S \rightarrow G_2 \rightarrow M \rightarrow G_1$$

Cell-Cycle in Yeast - Transitions Tree

- Each node represents a vector of the states.



transition tree for the main "attractor"

1764 = 86% of initial vectors.

(there are 6 other, smaller trees)

Cell-Cycle in Yeast - Paper Conclusions

The paper concludes qualitative* characteristics of the network:

- 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.

* As opposed to quantitative

Simulation of the Yeast Cell Cycle

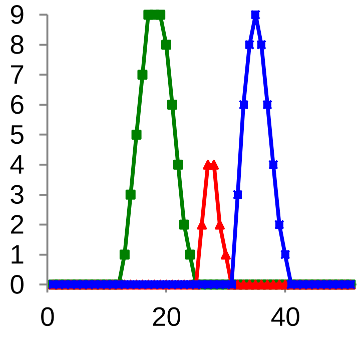
```
print("Yease cell cycle:")
nodes = ['Cln3','MBF','SBF', ...    ]

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

Simulation of the Yeast Cell Cycle (output)

```
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]
```

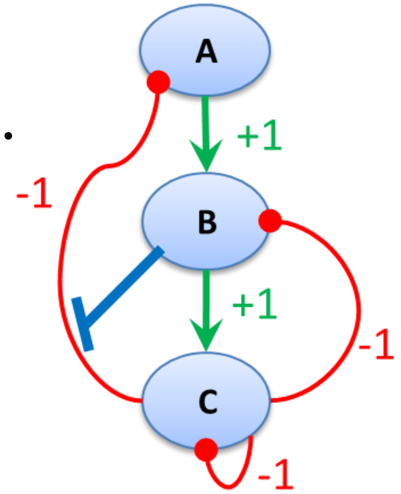
Extensions to the Model

- Discrete space: instead of 0/1, nodes can assume states between 0,…,U (e.g. U=9)



- Instead states changing by ±1, we may prefer the change to be a function of the total effect $\sigma = \sum_j w(j,i) \cdot s_j(t)$

- A new type of interactions: nodes may block edges.



- Delay on edges

S.Cerevisiae Cell Cycle Study

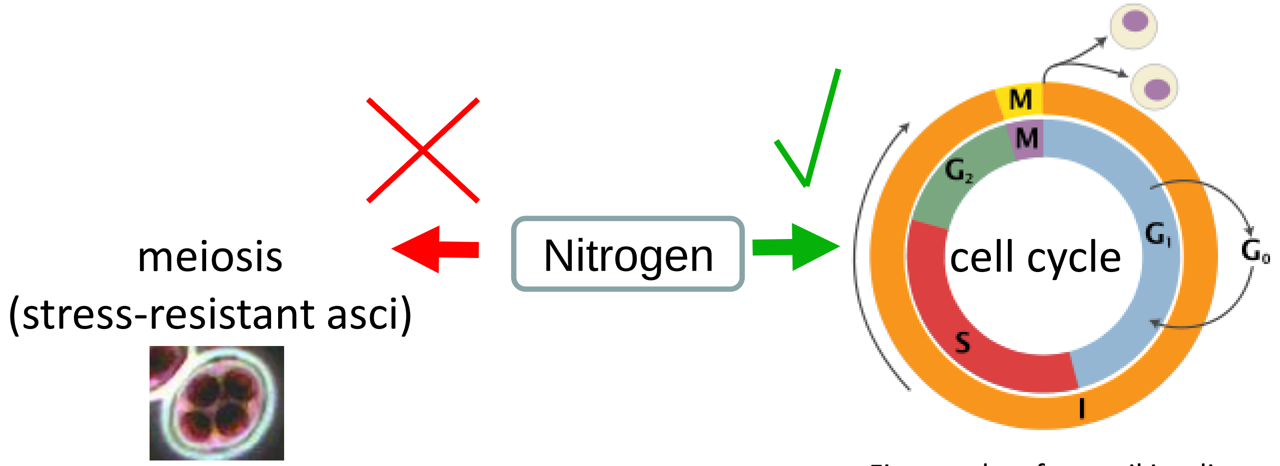
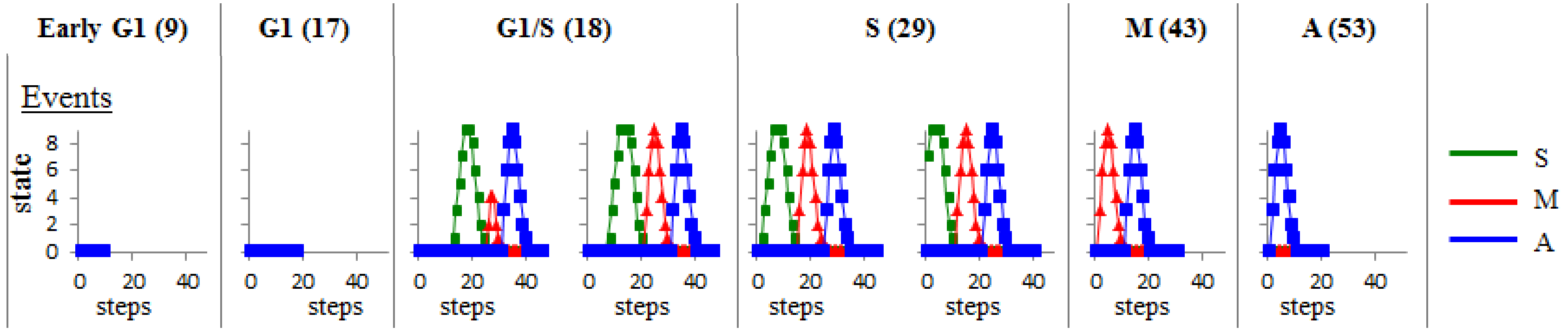


Figure taken from wikipedia

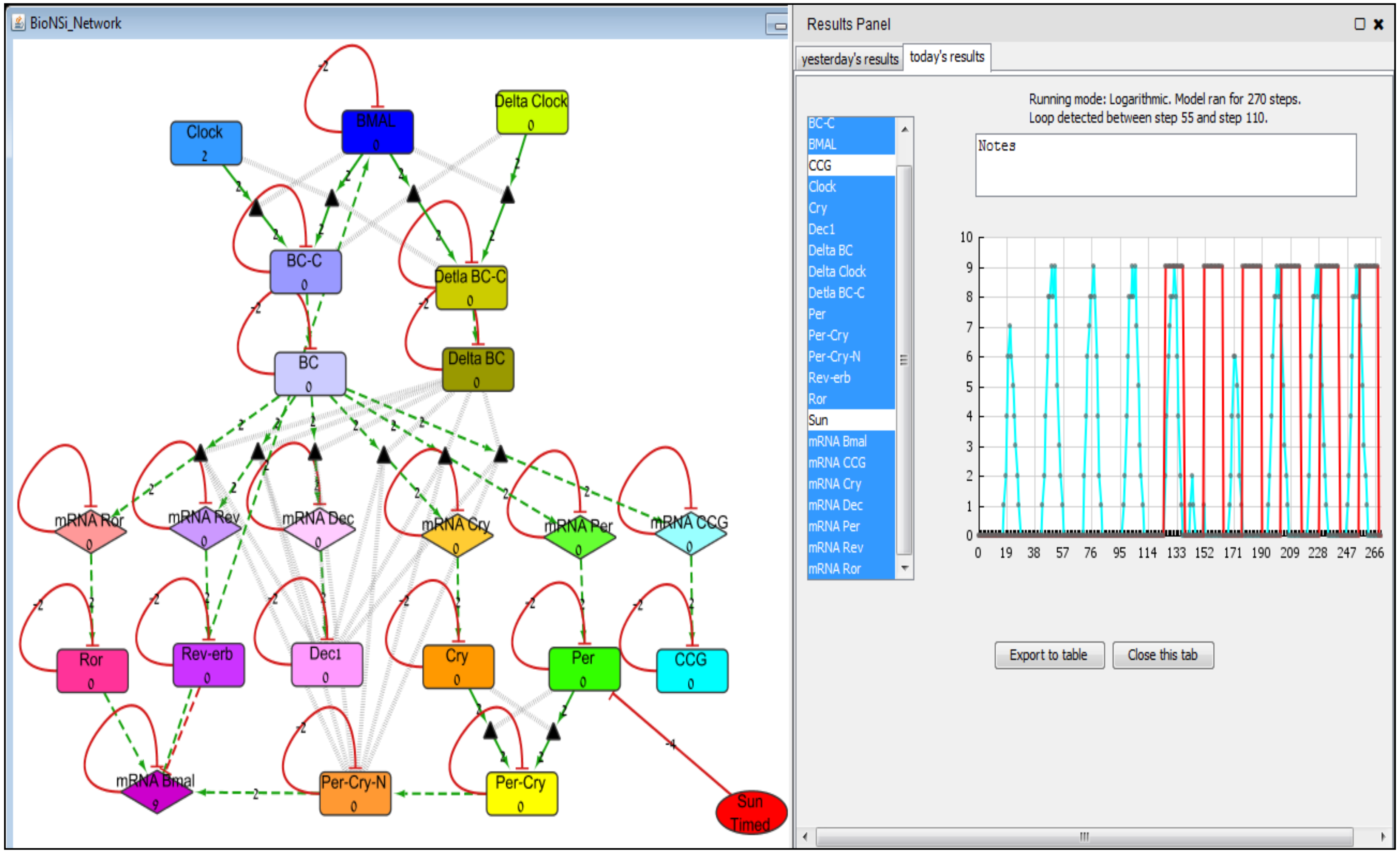


Cell cycle interrupts (α-factor) at different stages of the cell cycle

שתי פרסומות וחזרנו...

BioNSi

A Cytoscape plugin for the analysis of biological network dynamics: http://bionsi.wix.com/bionsi


Running mode: Logarithmic. Model ran for 270 steps.
Loop detected between step 55 and step 110.

Notes

The circadian clock in mammalians in a day-night regime

Computational Thinking for Life Scientists

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

  ca4ls.wikidot.com

Reflection

- Today we focused on the notions of graphs, models and simulations.
- Graphs theory is a well-studied model for various types of interactions.
- Categorization of models:
  - Continuous vs. discrete (special case: Boolean or logic)
  - Quantitative vs. qualitative
  - Stochastic vs. deterministic
- The reliability and predictive capability of computer simulations depend on the validity of the model
- Discrete notions (such as graphs, strings, etc.) are highly underrepresented in life science curricula, where continuous notions and probability are taught more widely.

Appendix: Counting Attractor Size

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

```
import itertools

def attractor_size(G, attractor):
    ''' count how many initial vectors end up in the attractor '''
    cnt=0
    n = len(G)

    for states in itertools.product([0,1], repeat=n):
        states = list(states) #convert tuple-->list
        final = run(G, states)
        if final == attractor:
            cnt+=1
    return cnt
```

Cartesian product

Simulation of the Yeast Cell Cycle

```
print("Yease cell cycle:")
nodes = ['Cln3','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]
     ]



final = [0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0] #main attractor
print(attractor_size(G, final), "initial vectors end at", final)
1764 initial vectors end at [0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0]
```