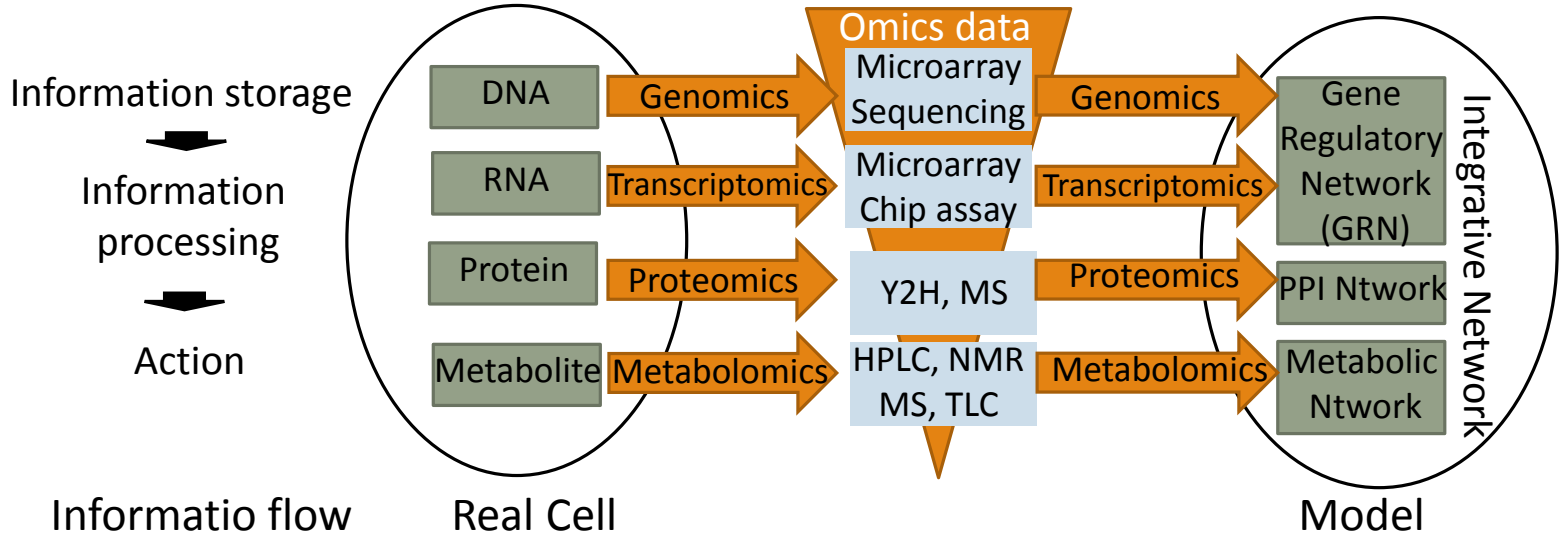
Biological Network analysis

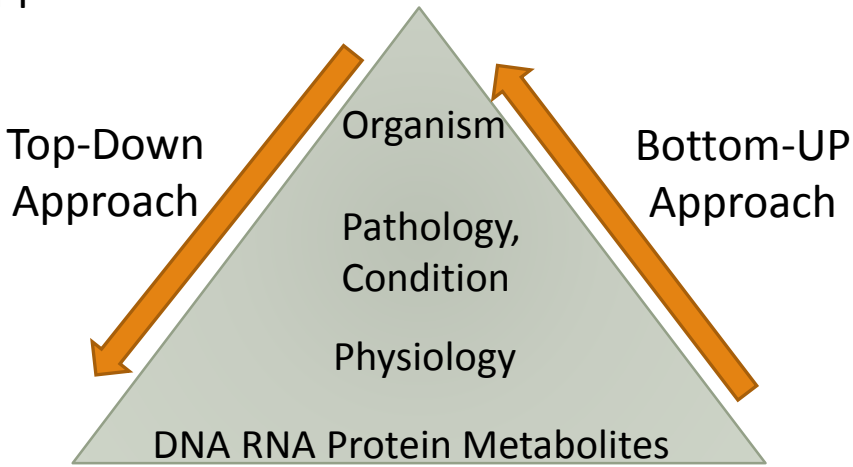
Gholamreza Bidkhori
Ph.D. In Bioinformatics



Why Network analysis in Biology

The central step toward a systems-level understanding of biology was to move away from *reductionist* to *wholist* approaches

make a snapshot of *all* elements at a certain level
By omics data



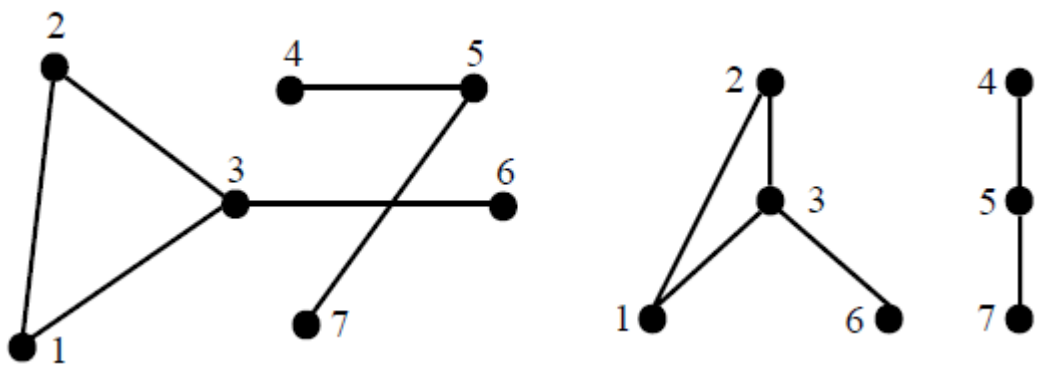
it is impossible to assemble an airplane by using a list of all parts like data generated by Omics methods

An introduction to graph theory

graphs

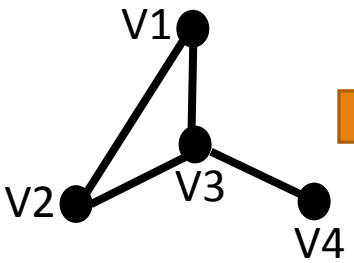
A *graph G* = (*V, E*) consists of a set of *vertices* (also called nodes or points) *V* and a set of *edges* (arcs, links) *E*, where each edge is assigned to two vertices
An edge e connecting the vertices u, v is denoted by {u, v}, we say
u and v are incident with e and adjacent (or neighbors) to each other. The vertices incident to an edge are called its **end-vertices**

Two graphical representations of the **undirected** graph *G* = (*V, E*)



V ={1, 2, 3, 4, 5, 6, 7}
edge set E = {{1, 2}, {2, 3}, {1, 3}, {3, 6}, {4, 5}, {5, 7}}

The *degree* of a vertex *v* is the number of edges that have *v* as end-vertex



	V1	V2	V3	V4
V1	0	1	1	0
V2	1	0	1	0
V3	1	1	0	1
V4	0	0	1	0

adjacency matrix
($n \times n$)

Upper triangle
Diagonal
lower triangle

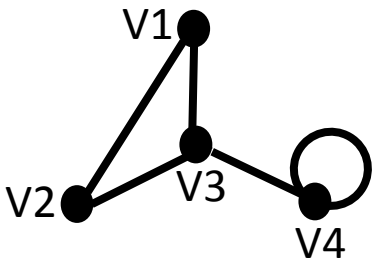
*L*1: ({v1, *v*2}, {v1, *v*3})
*L*2: ({v2, *v*1}, {v2, *v*3})
*L*3: ({v3, *v*1}, {v3, *v*2}, {v3, *v*4})
*L*4: ({v4, *v*3})

adjacency list

```
G=matrix(c(0,1,1,0,
 +          1,0,1,0,
 +          1,1,0,1,
 +          0,0,1,0),nrow=4,ncol=4)
colSums(G)
 [1] 2 2 3 1
```

GRAPH REPRESENTATION

An edge where the two end-vertices are the same vertex is called a *loop*.
A *loop-free* graph does not contain loops.



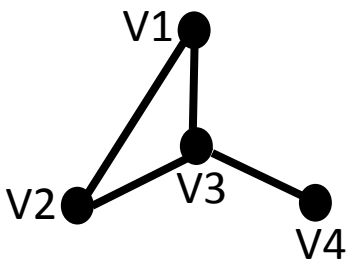
	V1	V2	V3	V4
V1	0	1	1	0
V2	1	0	1	0
V3	1	1	0	1
V4	0	0	1	1

adjacency matrix

Upper triangle
Diagonal
lower triangle

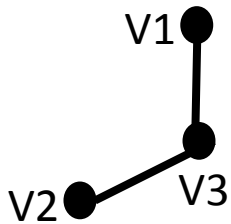
```
> G=matrix(c(0,1,1,0,
             1,0,1,0,
             1,1,0,1,
             0,0,1,1),nrow=4,ncol=4)
#Find degree
> colSums(G)
[1] 2 2 3 2
```

```
> s=diag(G) # Diagonal
> s
[1] 0 0 0 1
> if(all(s == 0)){
  print("loop-free graph")
  }else{
  print("loop found")
  }
[1] "loop found"
```

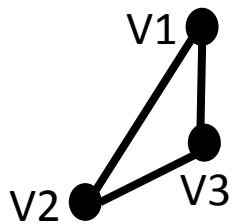


Graph

$G = (V, E)$



Subgraph

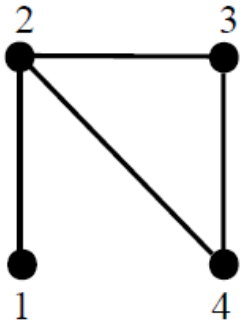


Induced subgraph

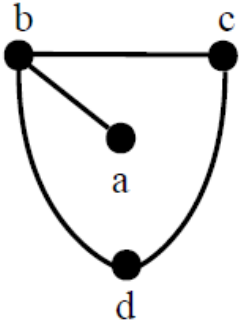
$G' = (V', E')$

```
> Induced_subgraph = G[-4,-4]
> subgraph=Induced_subgraph
> subgraph[1,2]=0
> subgraph[2,1]=0
> subgraph
     [,1] [,2] [,3]
[1,]    0    0    1
[2,]    0    0    1
[3,]    1    1    0
```

Two isomorphic graphs



$G_1 = (V_1, E_1)$



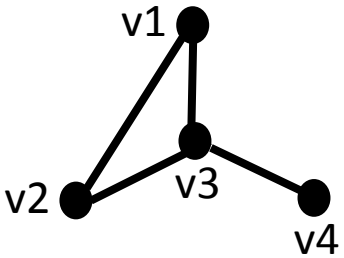
$G_2 = (V_2, E_2)$

$V_1 = \{1, 2, 3, 4\}$ and $E_1 = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{2, 4\}\}$

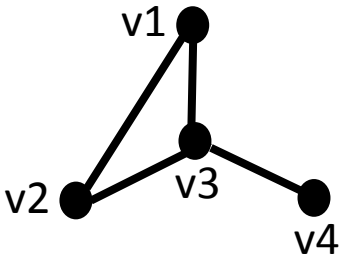
$V_2 = \{a, b, c, d\}$ and $E_2 = \{\{a, b\}, \{b, c\}, \{b, d\}, \{c, d\}\}$

```
all(G1 == G2)
 [1] TRUE
```

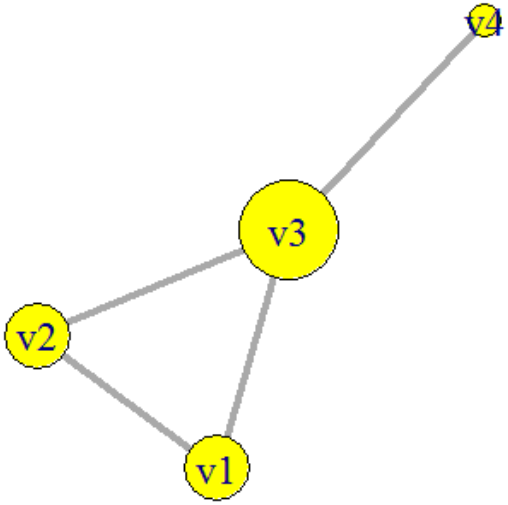
With different vertex and edge sets.




```
> data=data.frame(
  s=c("v1","v1", "v2", "v3"),
  t=c("v2","v3", "v3", "v4"))
#Library
> library(igraph)
# Turn it into igraph object
> G=graph_from_data_frame(data, directed=F)
> G
IGRAPH dcc0b94 UN-- 4 4 -
+ attr: name (v/c)
 + edges from dcc0b94 (vertex names):
[1] v1--v2 v1--v3 v2--v3 v3--v4
> V(G)
 + 4/4 vertices, named, from dcc0b94:
 [1] v1 v2 v3 v4
E(G)
 + 4/4 edges from dcc0b94 (vertex names):
 [1] v1--v2 v1--v3 v2--v3 v3--v4
# Count the number of degree for each node:
> d=degree(G, mode="all")
> d
 v1 v2 v3 v4
 2   2  3  1
```

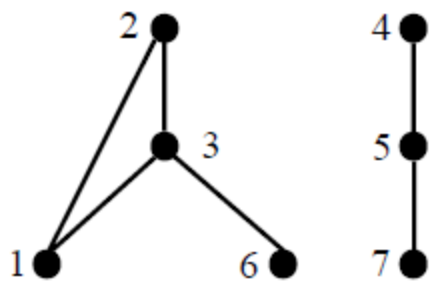



```
# Count the number of degree for each node:
> d=degree(G, mode="all", loops = TRUE)
> d
 v1 v2 v3 v4
 2   2  3  1
> d=degree(G, v = c("v1","v3"), mode="all")
> d
v1 v3
 2  3
# Plot
> plot(G, vertex.size=d*15,vertex.label.cex=1.5,
 vertex.color="yellow",edge.width=4)
```



A sequence ($v0, e1, v1, e2, v2, \ldots, vk-1, ek, vk$) of vertices and edges such that every edge ei has the end-vertices $vi-1$ and vi is called a *walk*
Usually the vertices are omitted and the walk is denoted by a sequence ($e1, e2, \ldots, ek$).
the walk *connects* $v0$ with vk and call $v0$ and vk the start- and end-vertex of the walk, respectively
a path is a sequence of edges which connect a sequence of vertices which, by most definitions, are all distinct from one another and if additionally all vertices are distinct the walk is called a *simple path*

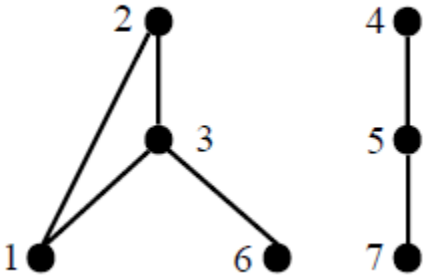
The *length* of a walk or path is given by its number of edges
A path with the same vertex as start- and end-vertex is a *cycle*
A graph without cycles is called an *acyclic graph*



the sequences:
({1, 2}, {2, 3}, {3, 6}, {6, 3}, {3, 1}) is a walk
({1, 2}, {2, 3}, {3, 1}) is a path and cycle

Two vertices of a graph are called *connected* if there exists a walk between them.
If any pair of different vertices of the graph is connected, the graph is *connected*.

A *connected component* of a graph *G* is a maximal connected subgraph of *G*.

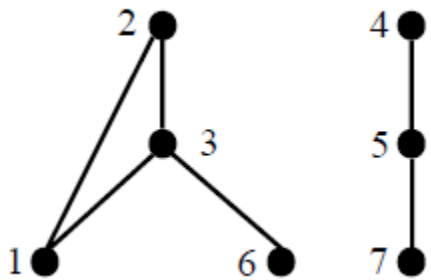


Two connected components.

```
> data=data.frame(
  s=c("v1","v1", "v2", "v3","v4","v5"),
  t=c("v2","v3", "v3", "v6","v5","v7"))
> G=graph_from_data_frame(data, directed=F)

#count and find connected component
> com=components(G)
> com$no
 [1] 2
> com$csize
 [1] 4 3
> groups(com)
 $`1`
 [1] "v1" "v2" "v3" "v6“
 $`2`
[1] "v4" "v5" "v7"
```

A *shortest path* between two vertices is a path with minimal length.
The *distance*
between two vertices is the length of a shortest path between them or ∞ if no such path exists.

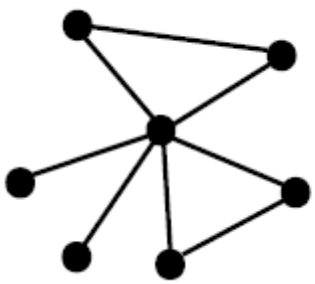


shortest path between vertex 1 and vertex 6:
path ({1, 3}, {3, 6})

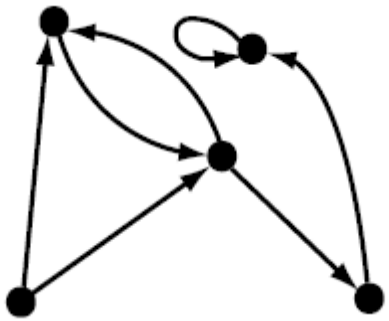
```
> shortest.paths(G)
     v1  v2  v3  v4  v5  v6  v7
v1    0   1   1 Inf Inf   2 Inf
v2    1   0   1 Inf Inf   2 Inf
v3    1   1   0 Inf Inf   1 Inf
v4  Inf Inf Inf   0   1 Inf   2
v5  Inf Inf Inf   1   0 Inf   1
v6    2   2   1 Inf Inf   0 Inf
v7  Inf Inf Inf   2   1 Inf   0
```

```
get.shortest.paths(G, "v1", mode = "all")
```

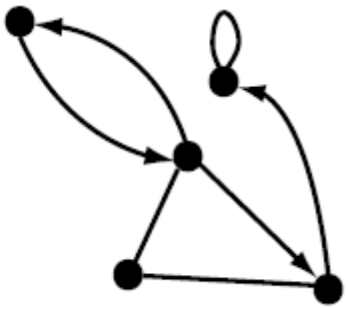
Undirected graph



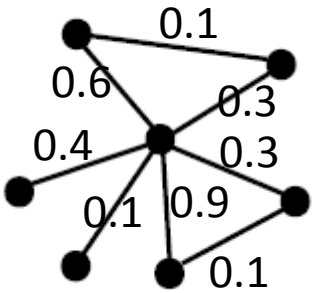
directed graph



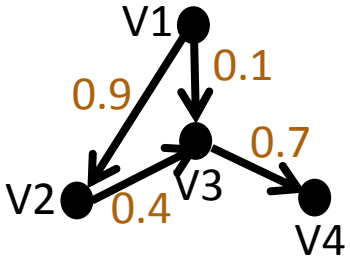
mixed graph

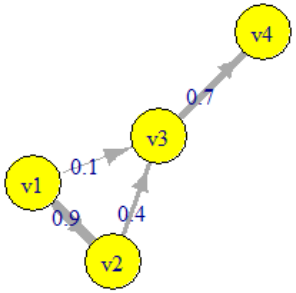


Weighted graph

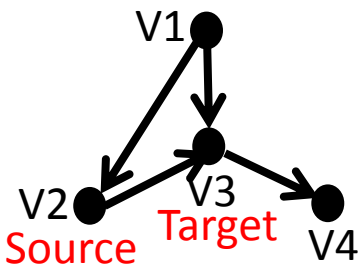


Weighted directed graph






```
data=data.frame(
 s=c("v1","v1", "v2", "v3"),
 t=c("v2","v3", "v3", "v4"),
 w=c(0.9,0.1,0.4,0.7))
> G=graph_from_data_frame(data, directed=T)
IGRAPH 99933d9 DN-- 4 4 --
attr: name (v/c), w (e/n)
edges from 99933d9 (vertex names):
[1] v1->v2 v1->v3 v2->v3 v3->v4
> E(G)$w
[1] 0.9 0.1 0.4 0.7
> plot(G, vertex.size=50, vertex.color="yellow",
edge.width=E(G)$w*8,vertex.label.cex=1.5,
edge.label=E(G)$w)
```

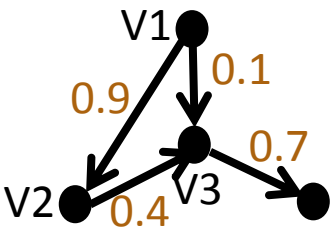



```
#Calculate indegree
d=degree(G, mode="in")
d
v1 v2 v3 v4
 0  1  2  1
#Calculate outdegree
> d=degree(G, mode="out")
```

```
> d
v1 v2 v3 v4
2  1  1  0
#Calculate in+outdegree
> d=degree(G, mode="all")
> d
v1 v2 v3 v4
2  2  3  1
```

walks, paths, and cycles are similar, but take the edge direction into account.
the walk ***strongly connects*** *v*0 with *vk* if the edge direction is taken into account.
Two vertices of a graph are called ***strongly connected*** if there exists such a walk between them.
If any pair of different vertices of the graph is strongly connected, the **graph is *strongly connected***.
A ***strongly connected component*** of a graph *G* is a maximally strongly connected subgraph of *G*.

Shortest (directed or undirected) paths between vertices

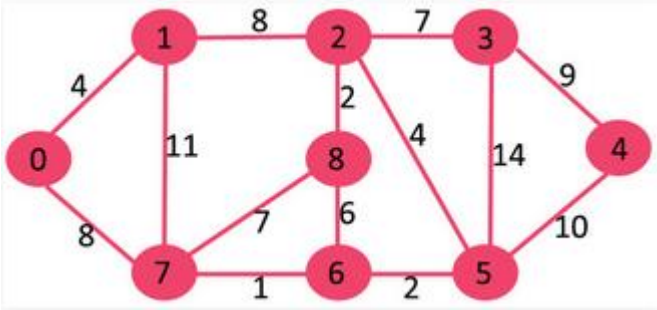


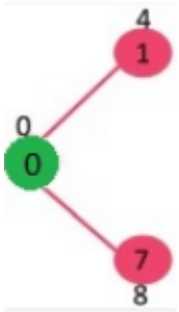

```
distances(G, v = "v1", mode = c("out"),
 weights = E(G)$w, algorithm = c( "dijkstra"))
 v1 v2 v3 v4 v1
 0 0.9 0.1 0.8
> distances(G, v = "v1", mode = c("in"),
weights = E(G)$w, algorithm = c( "dijkstra"))
 v1 v2 v3 v4 v1
 0 Inf Inf Inf
> distances(G, v = "v1", mode = c("all"),
weights = E(G)$w, algorithm = c( "dijkstra"))
v1 v2 v3 v4 v1
 0 0.5 0.1 0.8
```

If there are no weights, then an unweighted **breadth-first search** is used,
if all weights are positive, then **Dijkstra's algorithm** is used.
If there are negative weights and we do the calculation for more than 100 sources, then **Johnson's algorithm** is used. Otherwise the **Bellman-Ford algorithm** is used.

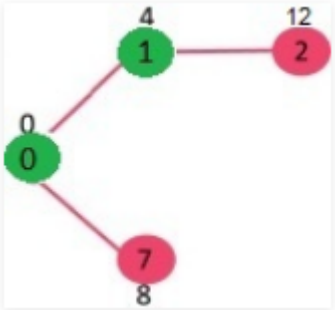
Dijkstra's shortest path algorithm

Let us understand with the following example:

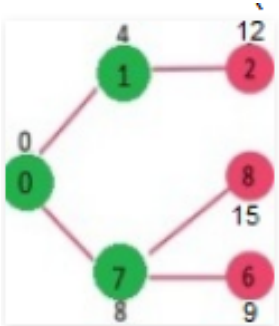




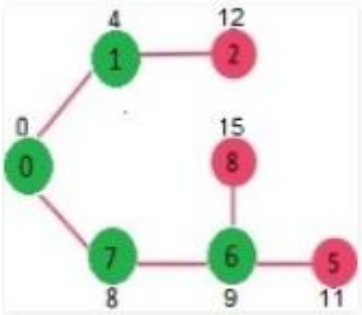
{0}



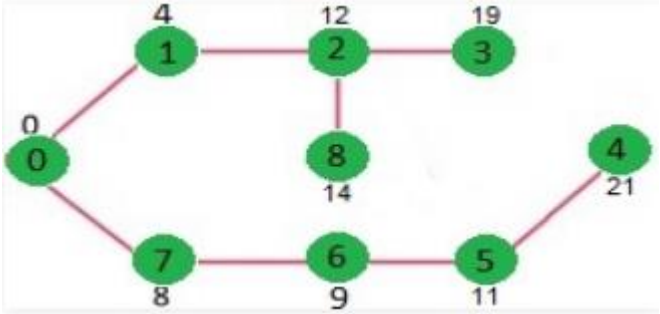
{0, 1}



{0, 1, 7}

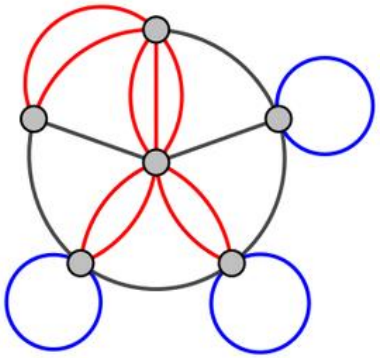


{0, 1, 7, 6}



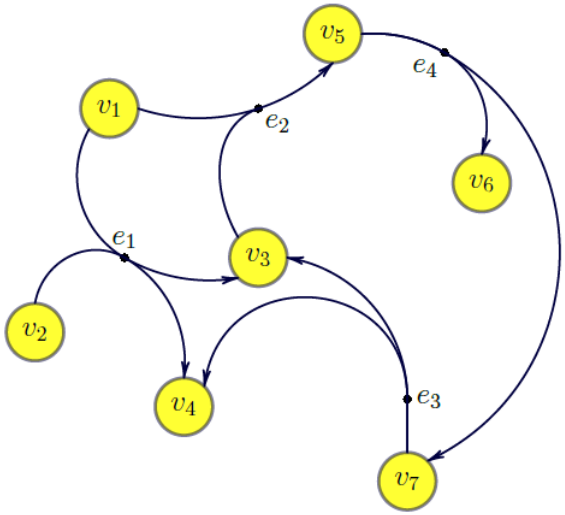
SPT (Shortest Path Tree) are shown in green colour.

Multigraphs are graphs containing multiple edges two or more edges that between two vertices in case of directed graphs have the same direction

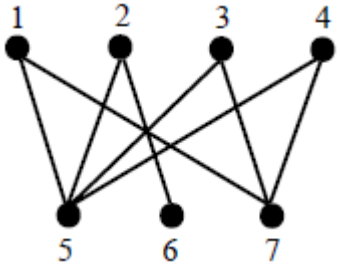


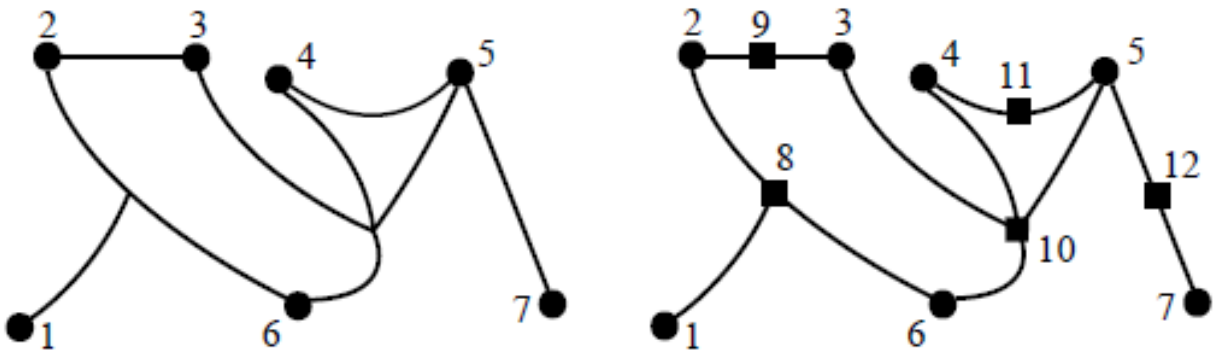
Undirected, loop-free graphs without multiple edges are called ***simple graphs***

hypergraph $G = (V, E)$ consists of a set of *vertices* V and a set of *hyperedges* E

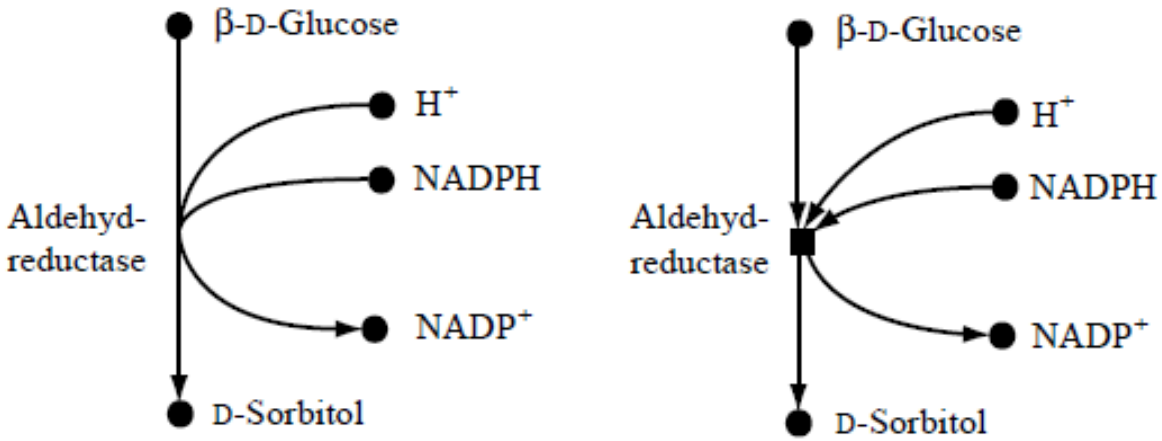


A graph $G = (V, E)$ is called ***bipartite*** if there is a partition of its vertex set $V = S \cup T$ such that each edge in E has exactly one end-vertex in S and one end-vertex in T





The hypergraph $G = (V, E)$ with vertex set $V = \{1, 2, 3, 4, 5, 6, 7\}$ and hyperedge set $E = \{\{1, 2, 6\}, \{2, 3\}, \{3, 4, 5, 6\}, \{4, 5\}, \{5, 7\}\}$ and its corresponding bipartite graph.
The two vertex sets S and T are represented by dots and squares, respectively.



A metabolic network and its modeling as bipartite graph.

A ***tree*** is an undirected, connected, acyclic graph.

The vertices of a tree with degree 1 are its ***leaves***, all other vertices are ***inner vertices***.

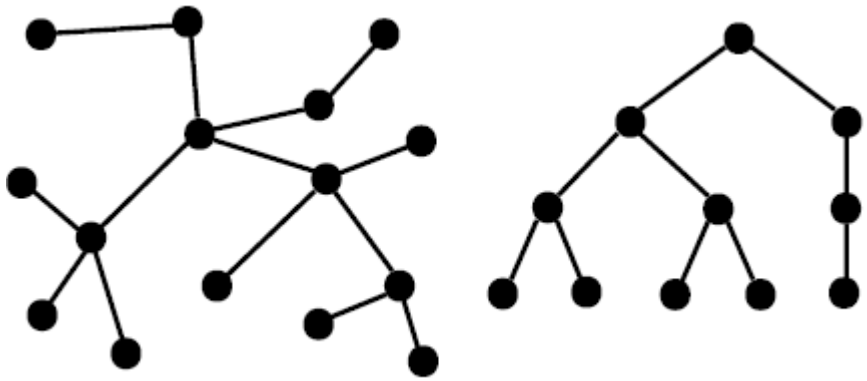
A ***rooted tree*** *(regarded* directed graph*)* consists of a tree $G = (V, E)$ and a distinguished vertex $r \in V$ called the *root*.

The ***depth*** of a vertex is the length of the path between the root and this vertex, the ***height*** of a tree is the maximum depth of a vertex.

A ***binary tree*** is a tree where each vertex has at most degree 3.

For a directed tree $G = (V, E)$ and an edge $(u, v) \in E$, the vertex u is the ***parent*** of v and v is the ***child*** of u.

For a connected, undirected graph G, a special tree can be computed, the ***spanning tree*** T of G. The spanning tree T is composed of all the vertices of G and a minimal set of edges that connect all vertices

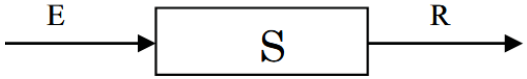


Biological networks

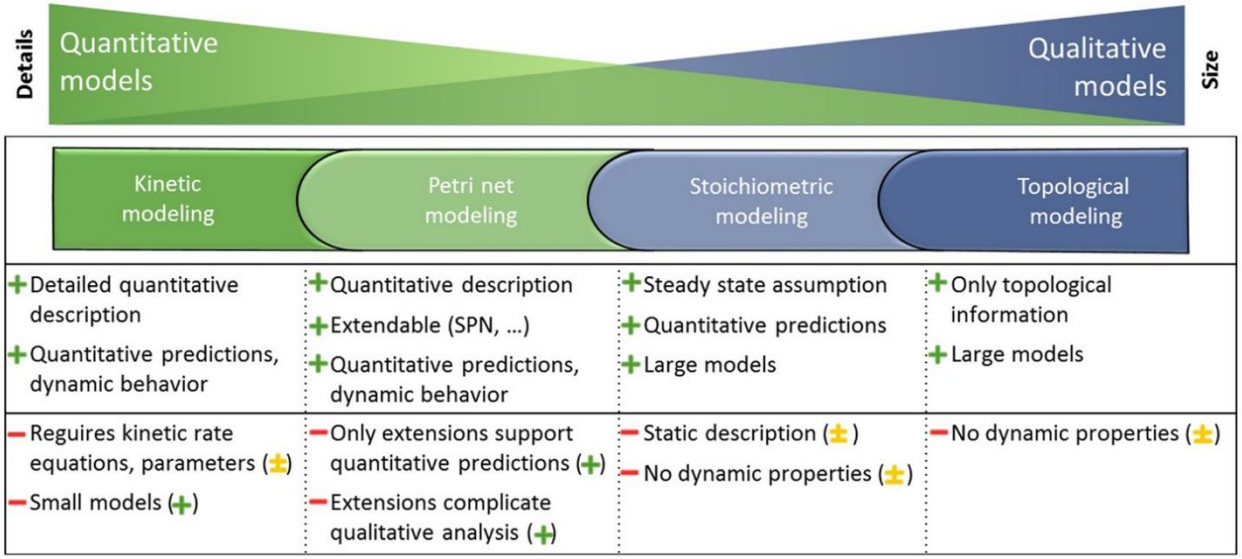
Type of Modeling of Biological Network

Static models Qualitative, Graph Representation, investigating the structure of

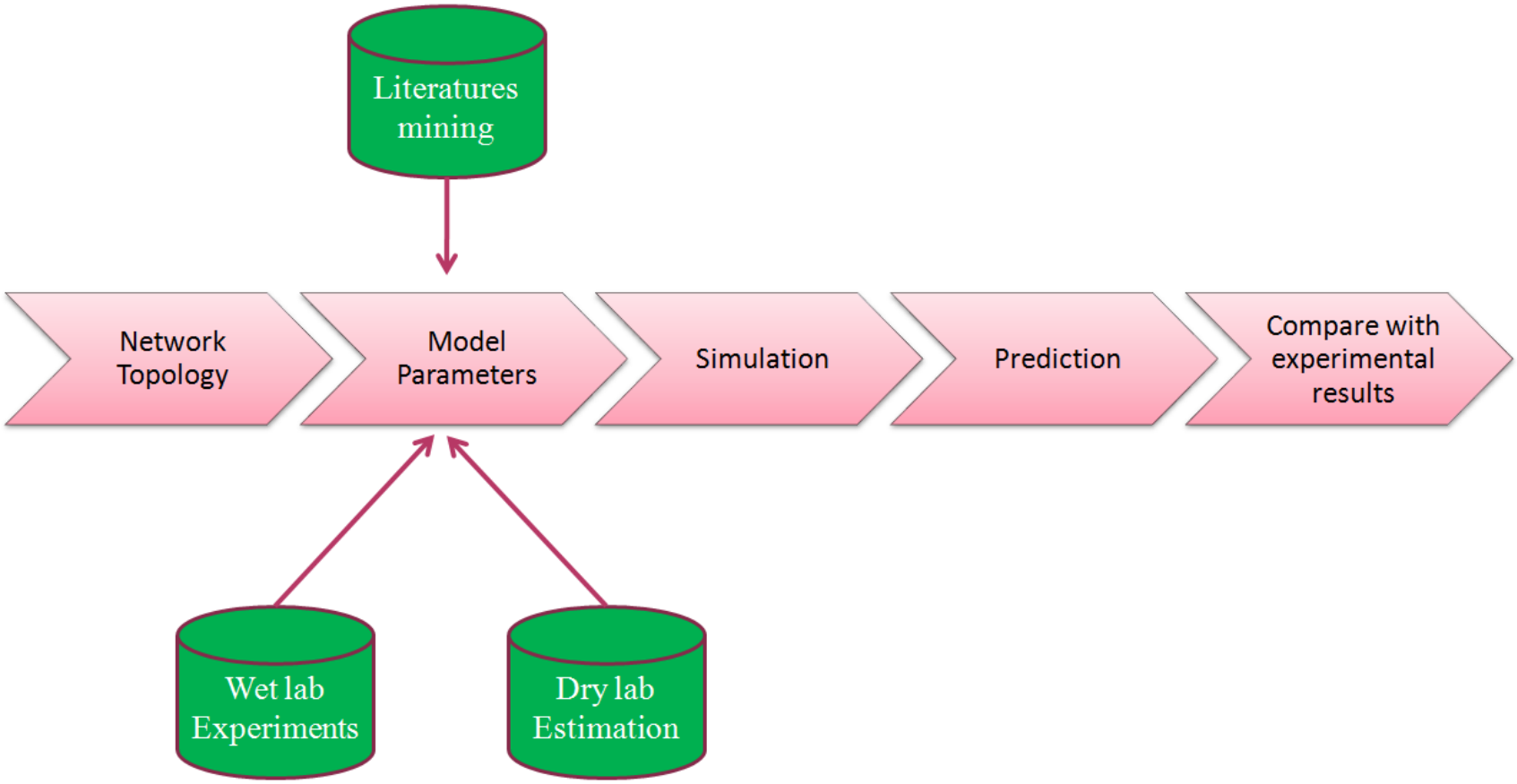
Dynamic models quantitatively modeling with rate laws for every step (over the time)



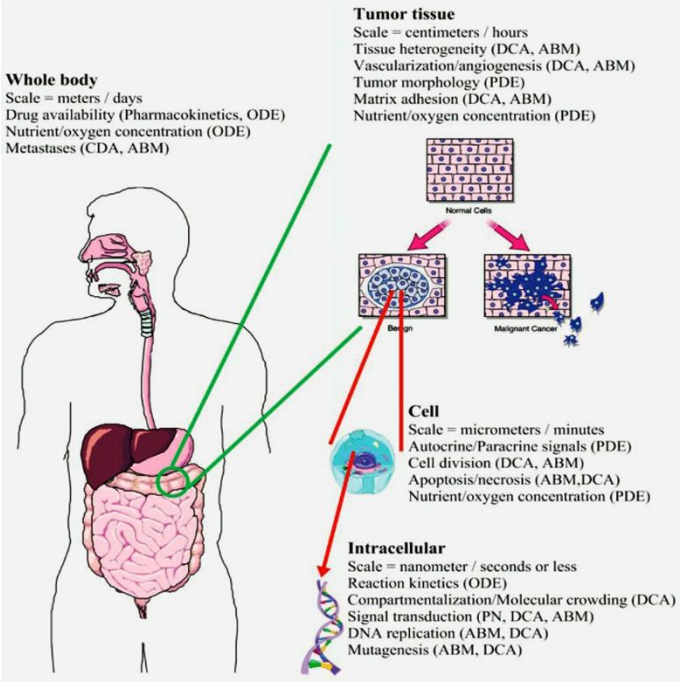
Given	To Find	Uses of Models
E and R	S	Understand
E and S	R	Predict
S and R	E	Control



Flow chart of the steps included in the preparation of a computational model



The scale of Biological Networks



Events: gene regulatory networks, metabolic networks, signal transduction etc.
Modeling methods: mainly ODE, Boolean network and petri net

Microscopic modeling

Mesoscopic modeling

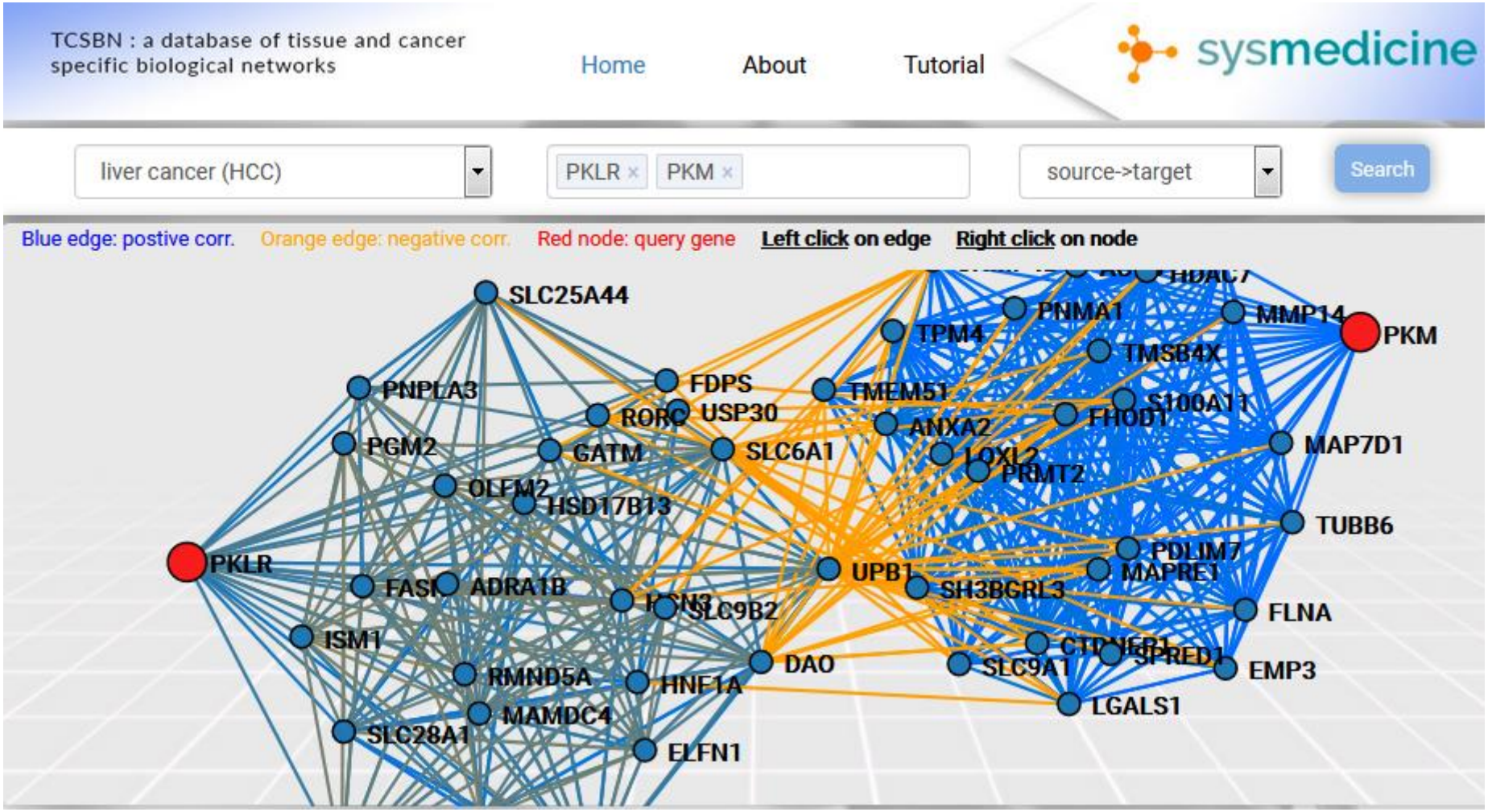
Macroscopic modeling

Events: cell death, division, growth, differentiation, migration and transformation in tissue, cell-cell and cell-matrix interactions
Modeling methods: mainly cellular automata, lattice-based and agent-base systems and

Events: diffusion of growth factors (EGF, PDGF, IGF and etc.), nutrients, O_2, CO_2 and etc. in tissue, angiogenesis, tissue patterning and invasion
Modeling methods: mainly reaction/diffusion systems, PDE, ABM and hybrid approaches

Correlation Network

http://inetmodels.com/



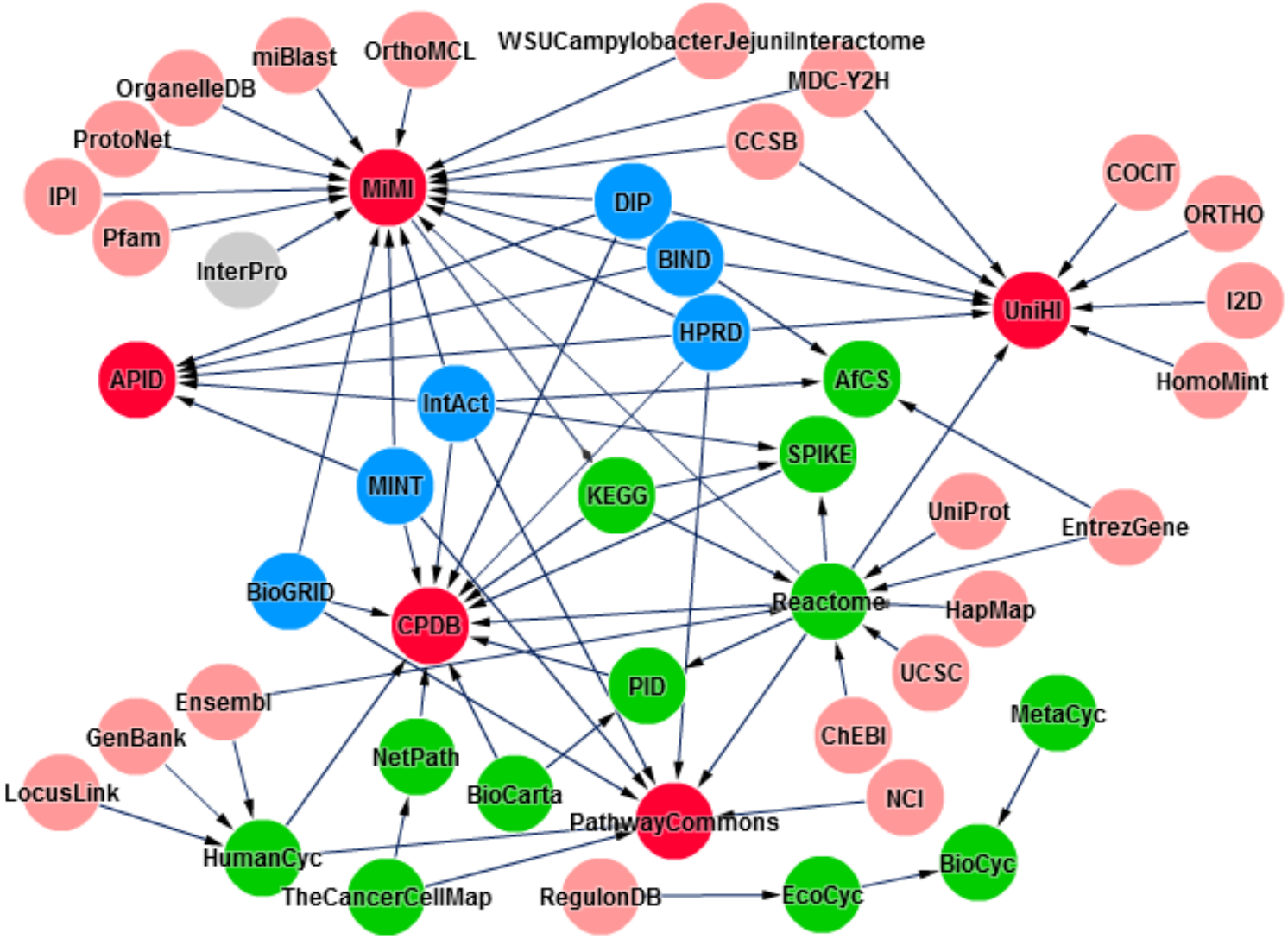
Pathway and interaction databases

selected data resources and databases for systems biology research

Pathway database	
KEGG	http://www.genome.jp/kegg/
Reactome	http://www.reactome.org
Recon X	http://humanmetabolism.org/
BioCyc	http://biocyc.org/
Pathway interaction database (PID)	http://pid.nci.nih.gov/
BioCarta	http://www.biocarta.com/
IntAct	http://www.ebi.ac.uk/intact/
Database of Interacting Protein (DIP)	http://dip.doe-mbi.ucla.edu/dip/Main.cgi

STRING https://string-db.org/

Pathway and interaction databases

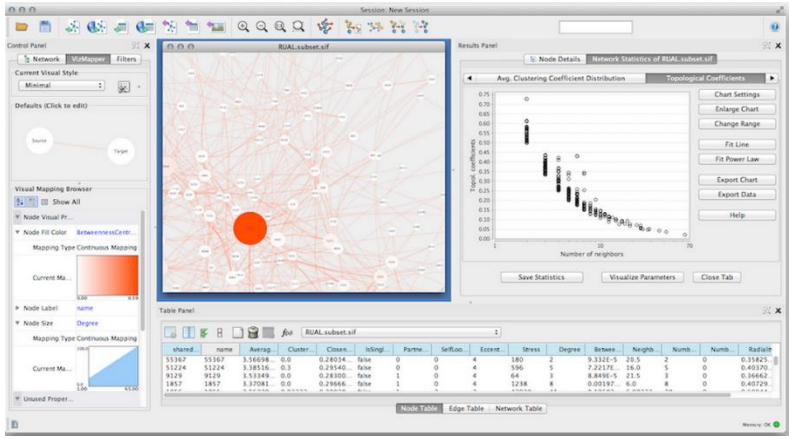


Overlap between important network and pathway databases and their interaction

Visualization tools

Edinburgh Pathway Editor
Cytoscape
BioUML
geWorkbench
Medusa
VANTED
BioTapestry

For several kinds of network manipulations, there are many Cytoscape plugins



PROPERTIES OF BIOLOGICAL NETWORKS

small world networks
scale-free networks
follow a scale-free power-law distribution
many vertices have few links, while there are some that are highly connected.

Protein Protein interaction

Database of Interacting Proteins

Archiving and validating experimentally determined protein-protein interactions

DIP Data | DIP Tools | About DIP | News | Resources | Preferences

Contribute to DIP!
Submit your interactions

View interaction networks
Get software

Get all interactions in DIP
Download DIP data

SEARCH DIP

Search DIP Help; Tutorial

Enter names, keywords, or identifiers associated with a protein, gene, interaction, or publication. (See all identifier types recognized by DIP.)

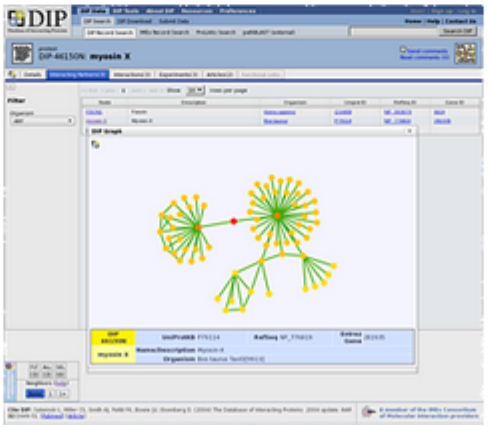
For more search options, try DIP Search

Search Help

Example DIP searches:

- Protein name: MVP
- Gene name: SUP35
- Keywords: yeast actin
- UniProt ID: P52960
- PubMed ID: 8676499
- PDB ID: 1PQS

Welcome



Welcome to the new and, hopefully, improved DIP Database site. While retaining most of the functionalities of the legacy site, the new portal utilizes modern web technologies to offer new graphical user interface while continuing tight integration with Cytoscape. Additional features include direct access to MIQL/PSICQUIC search engine site customization for registered users. Next few months will bring reimplementation of the remaining functionalities offered by the legacy site and fine-tuning of the caching subsystem which will make the site more responsive when dealing with large interaction datasets.

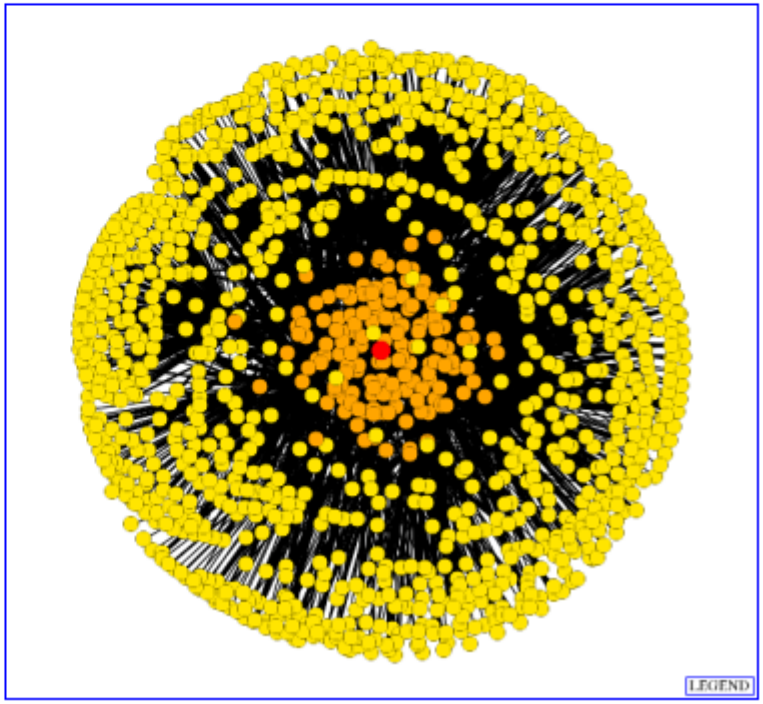
DIP NODE

DIP 368N	PIR	DNHU53	UniprotKB/SwissProt	P(
TP53	Name/Description	Cellular tumor antigen p53		
Organism	Homo sapiens			

Xrefs	GO Function	
D	Pfam:	PF08563, PF00870, PF07710
	InterPro:	IPR013872, IPR008967, IPR0(

DIP:368N (graph) - Mozilla Firefox

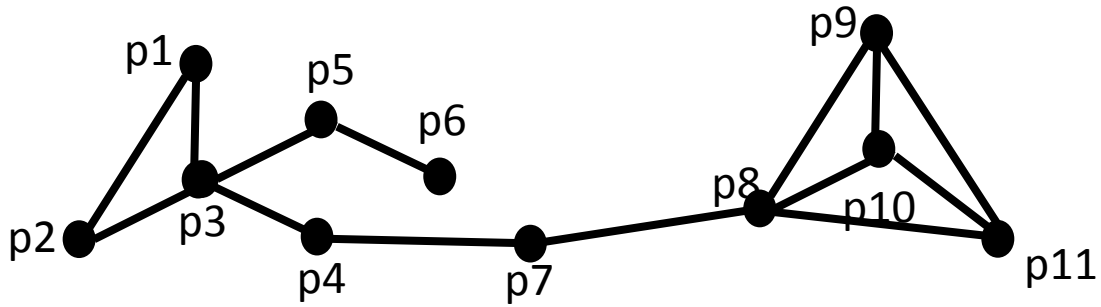
https://dip.doe-mbi.ucla.edu/dip/DIPgraph.cgi?PK=368&D=2



Comparison of the vertices to find node importance

Centrality Analysis

Centralities assign a real number to every vertex
They allow a pairwise comparison of the vertices
a vertex v1 is said to be more central or more important than a vertex v2 if C(v1) > C(v2).



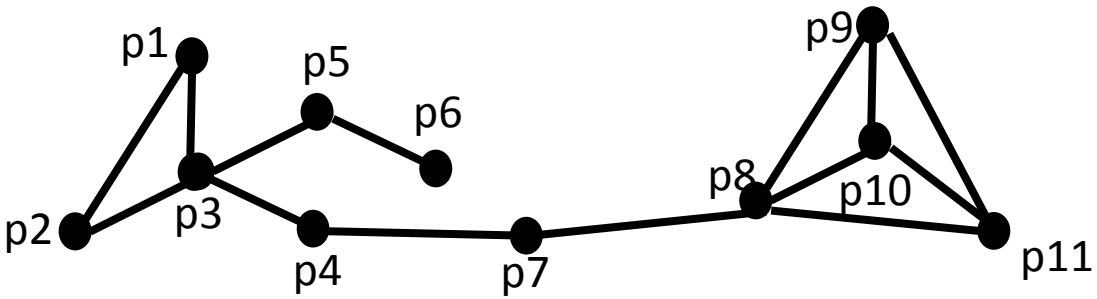
Definition 1 Degree centrality -> finding Hub

Definition 2 Eccentricity Centrality

$$\mathcal{C}_{\text{ecc}}(s) := \frac{1}{\max\{dist(s,t) : t \in V\}}$$

Let ***G* = (*V, E*)** be an undirected and connected graph
***dist* (*s,t*)** shortest path between *s* and *t*

Eccentricity uses information about the length of shortest paths between any two vertices of a network




```
> data=data.frame(
s=c("p1","p1","p2","p3","p3","p5","p4","p7","p8","p8","p8","p9","p9","p10"),
t=c("p2","p3","p3","p5","p4","p6","p7","p8","p9","p11","p10","p10","p11","p1"))
> G=graph_from_data_frame(data, directed=F)
```

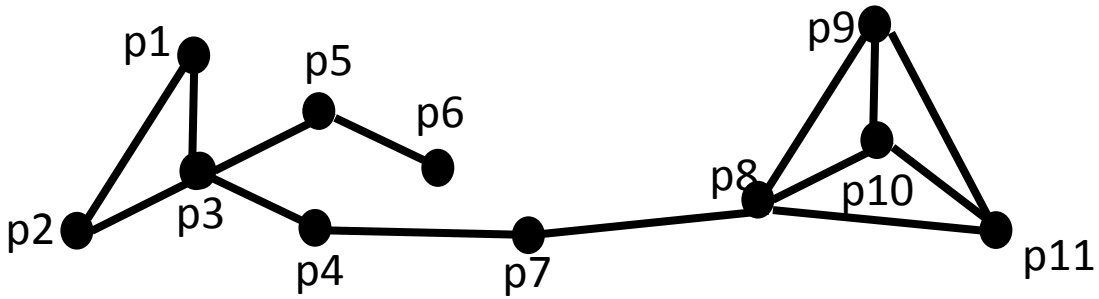
```
eccentricity(G, vids = V(G), mode = c("all"))
 p1 p2 p3 p5 p4 p7 p8 p9 p10 p6 p11
  5 5  4  5  3  4  5  6   6  6  6
sort(1/eccentricity(G, vids = V(G),
mode = c("all")),decreasing = T))
```

p4	0.333
p3	0.25
p7	0.25
p1	0.2
p2	0.2
p5	0.2
p8	0.2
p9	0.167
p10	0.16
p6	0.16
p11	0.16

Definition 3

Closeness Centrality

$C_{cl}(n) = 1 / avg(dist(n,m))$

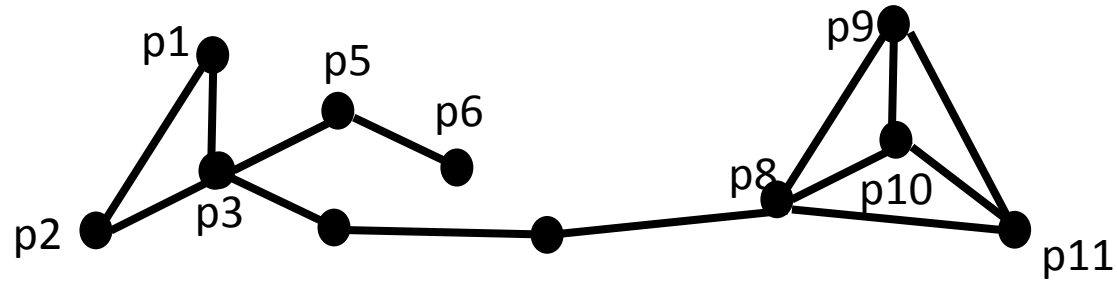



```
sort(closeness(G, vids = V(G), mode = c("all")),decreasing = T)
```

p4	0.045
p3	0.043
p7	0.043
p8	0.038
p5	0.033
p1	0.032
p2	0.032
p9	0.030
p10	0.030
p11	0.030
p6	0.025

betweenness

The **betweenness centrality** $C_b(n)$ of a node n is computed as follows:



$$C_b(n) = \sum_{s \neq n \neq t} (L_{st}(n) / L_{st})$$

where ***s*** and ***t*** are nodes in the network different from ***n***
L_{st} denotes the number of shortest paths from *s* to *t*,
$L_{st}(n)$ is the number of shortest paths from ***s*** to ***t*** that ***n*** lies on

It is normalized based on by dividing by the number of node pairs excluding *n*: $(N-1)(N-2)/2$, where *N* is the total number of nodes in the connected component

```
sort(betweenness(G, v = V(G), directed = F,
 normalized = T),decreasing = T)
```

p3	0.622222
p4	0.555556
p7	0.533333
p8	0.466667
p5	0.2
p1	0
p2	0
p9	0
p10	0
p6	0
p11	0