Algorithms for Bioinformatics

2018/2019

Graphs and Biological Networks

Pedro G. Ferreira
[d**CC**] @ Faculty of Sciences University of Porto

Outline

- Motivation for Biological Networks
- Examples of Biological Networks
- Graphs: definitions and representation
- Network Measures
- Types of Networks
- Class MyGraph
- Exercises

Biological Networks

- Cells have many constituents (molecules of RNA, DNA, proteins …) with biological characteristics arising from the complex interplay of these elements.

- A key challenge of modern biology is to understand the complex web of interactions that confer and contribute to the structure and function of the cell.

- High throughput techniques such as microarray and sequencing has allowed to interrogate the state of a cell at a given instant. Techniques such as Y2H screening or Protein chips help to determine how and when the different molecules interact with each other.

  Y2H: Yeast 2 Hibrid Screen is a genetic approach for the identification of potential protein-protein interactions

  Protein chips: are similar to cDNA microarrays and allow to detect the binding activity of proteins.

Biological Networks

- The elements of the cell form networks that are significantly different from random networks. Examples of cell networks include:
  - Protein-protein interactions;
  - Metabolic;
  - Signalling and Metabolic networks;
  - Protein phosporylation*;
  - Genetic interactions;
  - Co-expression networks;
  - Protein-DNA interactions;

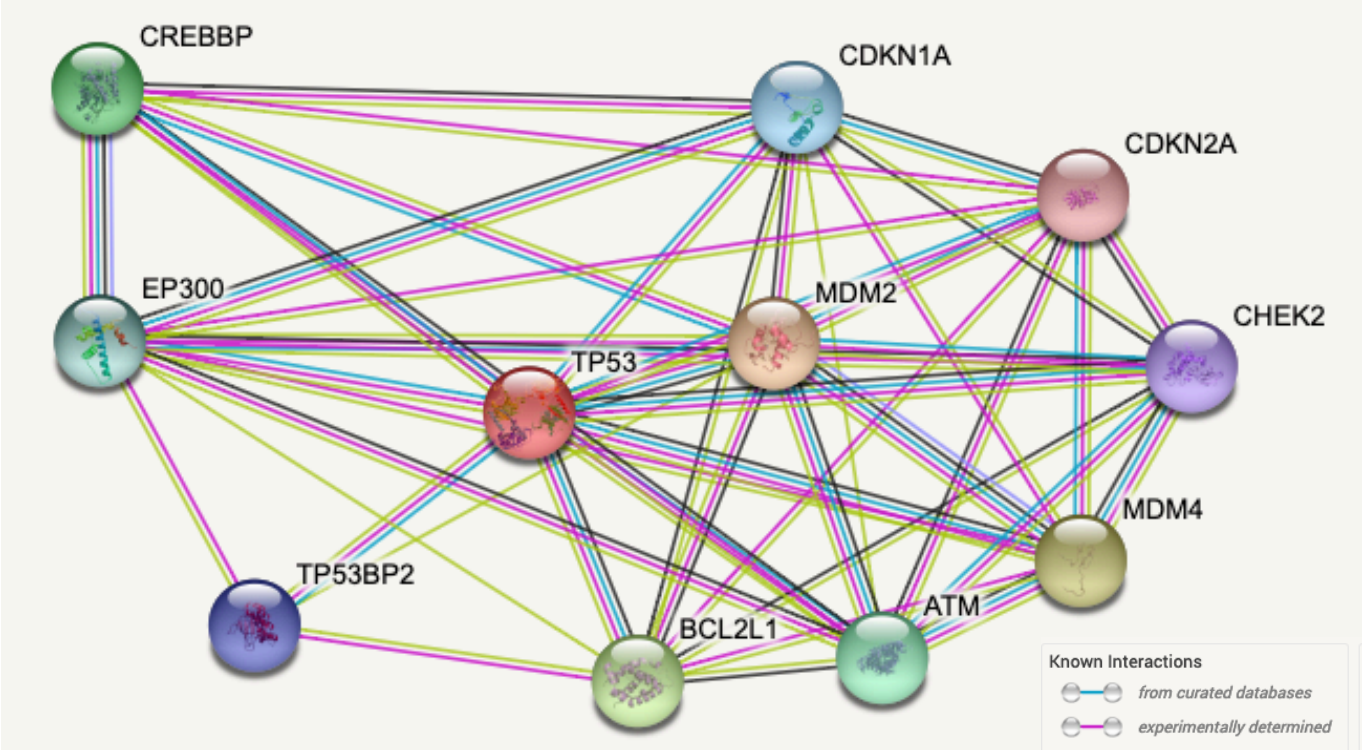
* Phosphorylation is a type of post-translational regulation where there is an addition of phosphoric group to the protein. This may turn on and off, altering their function and activity.

- The cell is formed by the interplay of networks and emerge as the sum of the interaction of its different elements. It is a *network of networks*.

- Networks of molecular interactions in the cell share characteristics that are similar to networks in other complex systems such as the internet, social media, society or computer chips.

Biological Networks: string-db.org

- Network information for gene TP53



Nodes:

Network nodes represent proteins

splice isoforms or post-translational modifications are collapsed, i.e. each node represents all the proteins produced by a single, protein-coding gene locus.

Edges:

Edges represent protein-protein associations

associations are meant to be specific and meaningful, i.e. proteins jointly contribute to a shared function; this does not necessarily mean they are physically binding each other.

Known Interactions

- *from curated databases*
- *experimentally determined*

Predicted Interactions

- *gene neighborhood*
- *gene fusions*
- *gene co-occurrence*

Others

- *textmining*
- *co-expression*
- *protein homology*

TP53 — *Cellular tumor antigen p53; Acts as a tumor suppressor in many tumor types; induces growth arrest or apoptosis depending on the physiological circumstances and cell type. Involved in cell cycle regulation as a trans-activator that acts to negatively regulate cell division by controlling a set of genes required for this process. One of the activated genes is an inhibitor of cyclin-dependent kinases. Apoptosis induction seems to be mediated either by stimulation of BAX and FAS antigen expression, or by repression of Bcl-2 expression. In cooperation with mitochondrial PPIF is involved in [...] (393 aa)*

Predicted Functional Partners:

		Neighborhood	Gene Fusion	Cooccurence	Coexpression	Experiments	Databases	Textmining	[Homology]	Score
MDM2	*E3 ubiquitin-protein ligase Mdm2; E3 ubiquitin-protein ligase that mediates ubiquitination of p53/TP53, leading to its degra...*					•	•	•		0.999
MDM4	*Protein Mdm4; Inhibits p53/TP53- and TP73/p73-mediated cell cycle arrest and apoptosis by binding its transcriptional act...*					•	•	•		0.999
BCL2L1	*Bcl-2-like protein 1; Potent inhibitor of cell death. Inhibits activation of caspases. Appears to regulate cell death by blockin...*						•	•		0.999
CREBBP	*CREB-binding protein; Acetylates histones, giving a specific tag for transcriptional activation. Also acetylates non-histone ...*					•	•	•		0.999
ATM	*Serine-protein kinase ATM; Serine/threonine protein kinase which activates checkpoint signaling upon double strand break...*					•	•	•		0.999
EP300	*Histone acetyltransferase p300; Functions as histone acetyltransferase and regulates transcription via chromatin remodeli...*					•	•	•		0.999
CDKN1A	*Cyclin-dependent kinase inhibitor 1; May be involved in p53/TP53 mediated inhibition of cellular proliferation in response t...*					•	•	•		0.999
TP53BP2	*Apoptosis-stimulating of p53 protein 2; Regulator that plays a central role in regulation of apoptosis and cell growth via its ...*					•	•	•		0.998
CHEK2	*Serine/threonine-protein kinase Chk2; Serine/threonine-protein kinase which is required for checkpoint-mediated cell cycle ...*					•	•	•		0.998
CDKN2A	*Cyclin-dependent kinase inhibitor 2A; Acts as a negative regulator of the proliferation of normal cells by interacting strongl...*					•	•	•		0.997

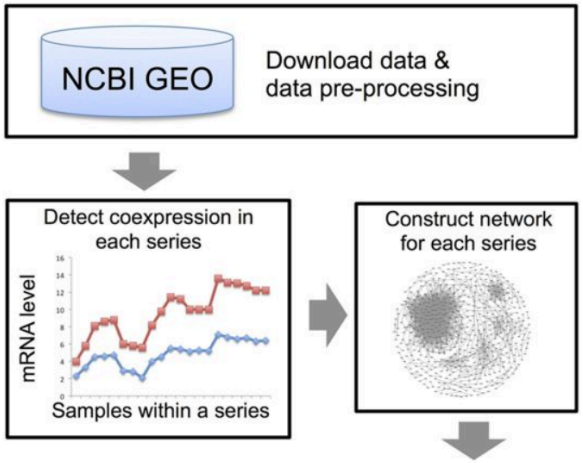
Network Stats

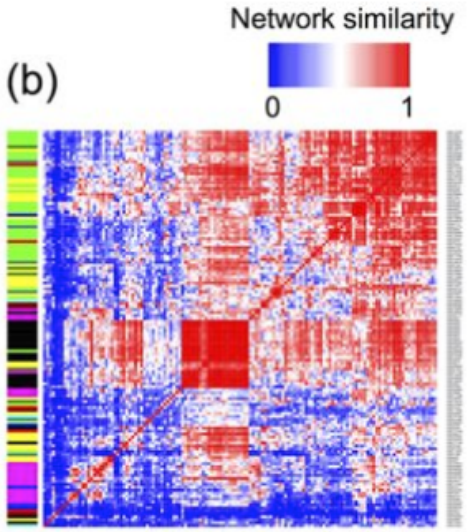
number of nodes:	11	expected number of edges:	22
number of edges:	46	PPI enrichment p-value:	4.66e-06
average node degree:	8.36		
avg. local clustering coefficient:	0.91		

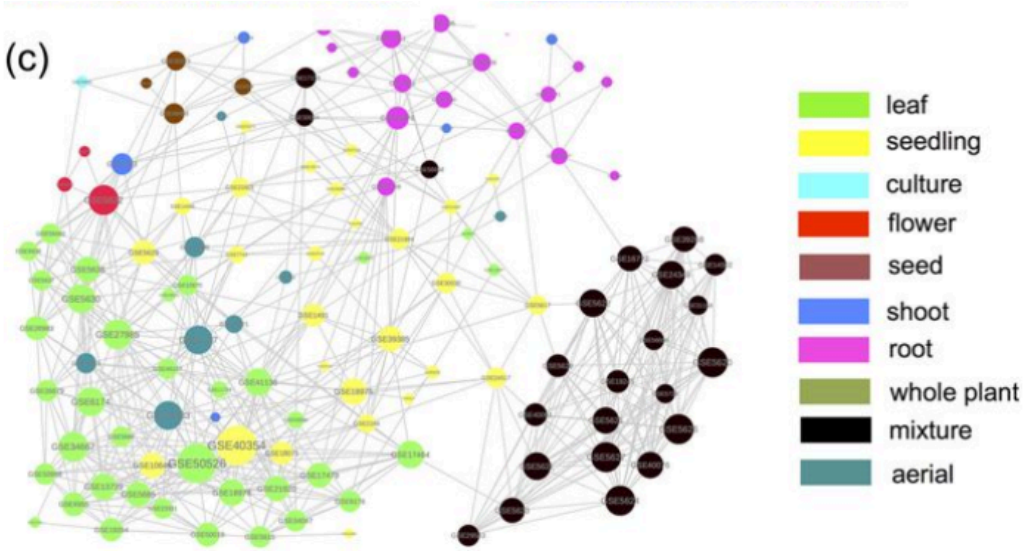
your network has significantly more interactions than expected (what does that mean?)

Biological Networks

Figures adapted from: *Pan- and core-network analysis of co-expression genes in a model plant*
Fei He & Sergei Maslov. Scientific Reports volume 6, Article number: 38956 (2016)

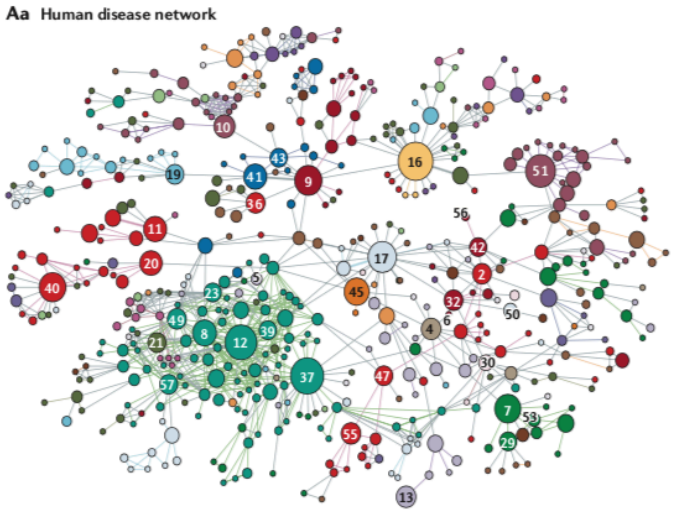




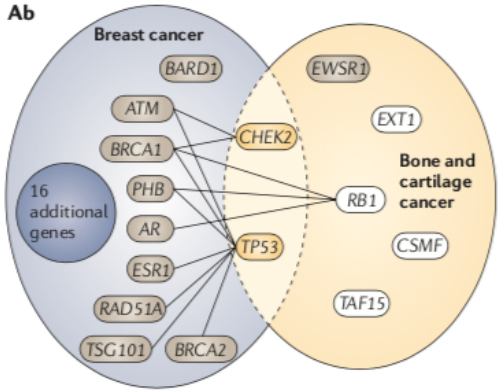


Biological Networks

Figures adapted from: *Network medicine: a network-based approach to human disease*
Albert-László Barabási, Natali Gulbahce and Joseph Loscalzo. Nature Reviews Genetics, 12, 2011.



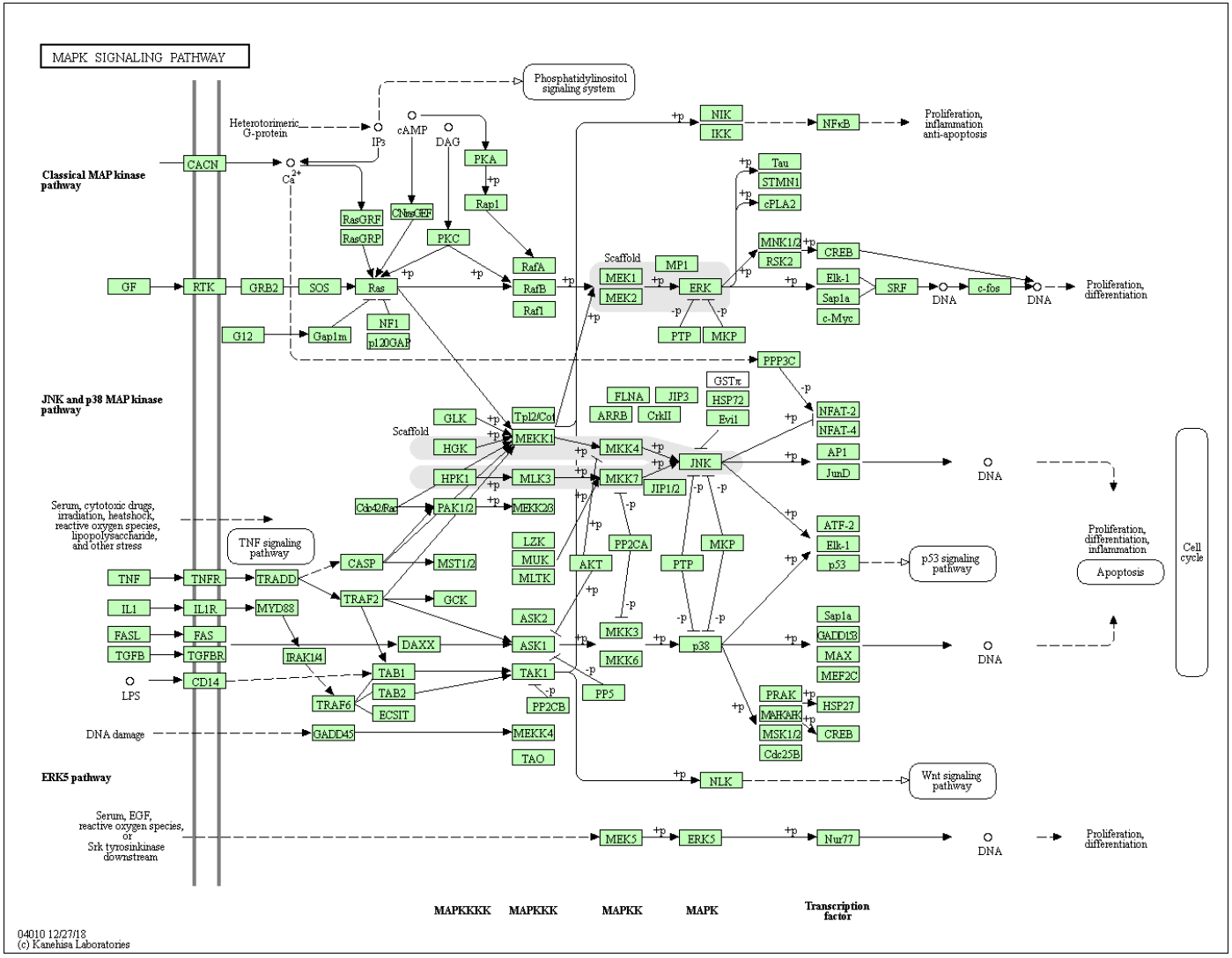
1 Aldosteronism
2 Alzheimer's disease
3 Anaemia, congenital deserythropoietic
4 Asthma
5 Ataxia-telangiectasia
6 Atherosclerosis
7 Blood group
8 Breast cancer
9 Cardiomyopathy
10 Cataract
11 Charcot–Marie–Tooth disease
12 Colon cancer
13 Complement component deficiency
14 Coronary artery disease
15 Coronary spasm
16 Deafness
17 Diabetes mellitus
18 Enolase-β deficiency
19 Epidermolysis bullosa
20 Epilepsy
21 Fanconi's anaemia
22 Fatty liver
23 Gastric cancer
24 Gilbert's syndrome
25 Glaucoma 1A
26 Goitre congenital
27 HARP syndrome
28 HELLP syndrome
29 Haemolytic anaemia
30 Hirschprung disease
31 Hyperbilirubinaemia
32 Hypertension
33 Hypertension diastolic
34 Hyperthyroidism
35 Hypoaldosteronism
36 Leigh syndrome
37 Leukaemia
38 Low renin hypertension
39 Lymphoma
40 Mental retardation
41 Muscular dystrophy
42 Myocardial infarction
43 Myopathy
44 Nucleoside phosphorylase deficiency
45 Obesity
46 Paraganglioma
47 Parkinson's disease
48 Pheochromocytoma
49 Prostate cancer
50 Pseudohypoaldosteronism
51 Retinitis pigmentosa
52 Schizoaffective disorder
53 Spherocytosis
54 Spina bifida
55 Spinocerebellar ataxia
56 Stroke
57 Thyroid carcinoma
58 Total iodide organification defect
59 Trifunctional protein deficiency
60 Unipolar depression



- Human gene disease networks where two diseases are linked if they share disease-associated genes.

Biological Networks KEGG

KEGG. he mitogen-activated protein kinase (MAPK) cascade is a highly conserved module that is involved in various cellular functions, including cell proliferation, differentiation and migration.



Graphs

- Graphs are mathematical concepts in which some of the pairs of objects in the set are related.

- Formally: G = <V, E>
  - V = vertices or nodes
  - E = edges or arcs, connecting a pair of nodes *(u,v)* indicating the existence of a connection between *u* and *v*.

- Graphs can be:
  - **Undirected** if edges are unordered
  - **Directed** or **digraph** if edges have an orientation, i.e. pairs are ordered
  - **Weighted** if numerical weights are associated to edges.

- Graphs have different computational representations depending on their nature (number of nodes, how connected they are, with or without weights, …) and the type of algorithms that will operate on the graph. Matrices or adjacency lists are the typical way to represent graphs.

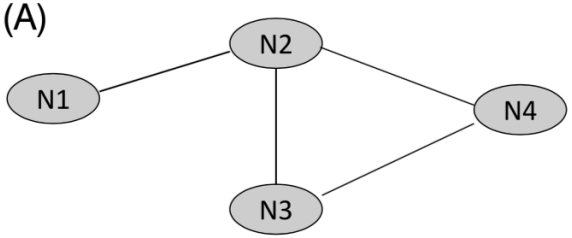
Graphs representation

- Matrices: all possible combination of vertices are represented.
  - Undirect graph: the rows and cols represent the nodes. *Cell(i,j)* represents an edge between node *i* and *j:* 1 if connected and 0 otherwise.
  - Direct graph: rows represent the origin node and cols the destination node. *Cell(i,j)* represents an edge between node *i* and *j:* 1 if connected and 0 otherwise.
  - Weighted graph: *Cell(i,j)* represents the weight of the edge between node *i* and *j.* 0 if not connected.

- Adjacency lists represent only the existing edges. Each vertex is linked to a list with associated neighbour nodes.
  - Direct graph: list for vertex *v* includes all destination nodes for the edges where *v* is the origin.
  - Undirect graph: the edge only exist for one of the directions.

Graphs

(A)

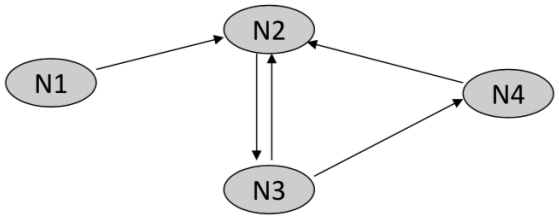


(B)

	N1	N2	N3	N4
N1	0	1	0	0
N2		0	1	1
N3			0	1
N4				0

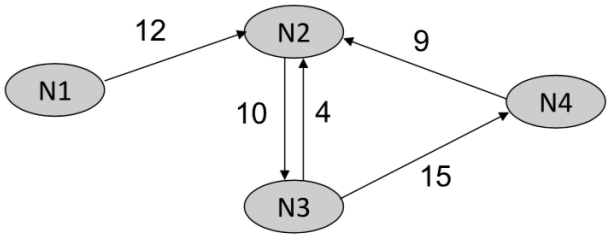
(C)

1 => [2]
2 => [3,4]
3 => [4]
4 => [ ]



	N1	N2	N3	N4
N1	0	1	0	0
N2	0	0	1	0
N3	0	1	0	1
N4	0	1	0	0

1 => [2]
2 => [3]
3 => [2,4]
4 => [2]



	N1	N2	N3	N4
N1	0	12	0	0
N2	0	0	10	0
N3	0	4	0	15
N4	0	9	0	0

1 => [(2,12)]
2 => [(3,10)]
3 => [(2,4),(4,15)]
4 => [(2,9)]

- From Bioinformatics Algorithms, Rocha & Ferreira

Class MyGraph

- Implement a Graph class for a direct graph with a representation as adjacency matrices using dictionaries. Keys are vertices and values are the list of adjacent nodes of the key node.

```
1. class MyGraph:
2.     def __init__(self, g = {}):
3.         ''''' Constructor -
    takes dictionary to fill the graph as input; default is empty dictionary '''
4.         self.graph = g
5.
6.     def print_graph(self):
7.         ''''' Prints the content of the graph as adjacency list '''
8.         for v in self.graph.keys():
9.             print (v, " -> ", self.graph[v])
10.
```

MyGraph class

- Implement the following functions on graphs:
  - **get_nodes**: retrieve list of nodes in graph
  - **get_edges**: retrieve list of edges in graph
  - **size**: retrieve the size (as tuple) of the graph as the number of nodes and number of edges
  - **add_vertex**: add vertex to graph (test if exists and only insert if not)
  - **add_edge**: add an edge between vertex *orig* and *dest*

MyGraph class


```
1.  def get_nodes(self):
2.      ''''' Returns list of nodes in the graph '''
3.      return list(self.graph.keys())
4.
5.  def get_edges(self):
6.      ''''' Returns edges in the graph as a list of tuples (origin, destinat
    ion) '''
7.      edges = []
8.      for v in self.graph.keys():
9.          for d in self.graph[v]:
10.             edges.append((v,d))
11.     return edges
12.
13. def size(self):
14.     ''''' Returns size of the graph : number of nodes, number of edges '''

15.     return len(self.get_nodes()), len(self.get_edges())
16.
```

MyGraph class

```
1. def add_vertex(self, v):
2.     ''''' Add a vertex to the graph; tests if vertex exists not adding
   if it does '''
3.     if v not in self.graph.keys():
4.         self.graph[v] = []
5.
6. def add_edge(self, o, d):
7.     ''''' Add edge to the graph; if vertices do not exist, they are add
   ed to the graph '''
8.     if o not in self.graph.keys():
9.         self.add_vertex(o)
10.    if d not in self.graph.keys():
11.        self.add_vertex(d)
12.    if d not in self.graph[o]:
13.        self.graph[o].append(d)
14.
```

MyGraph class

- Get successors, predecessors and adjacent vertices of a given vertex

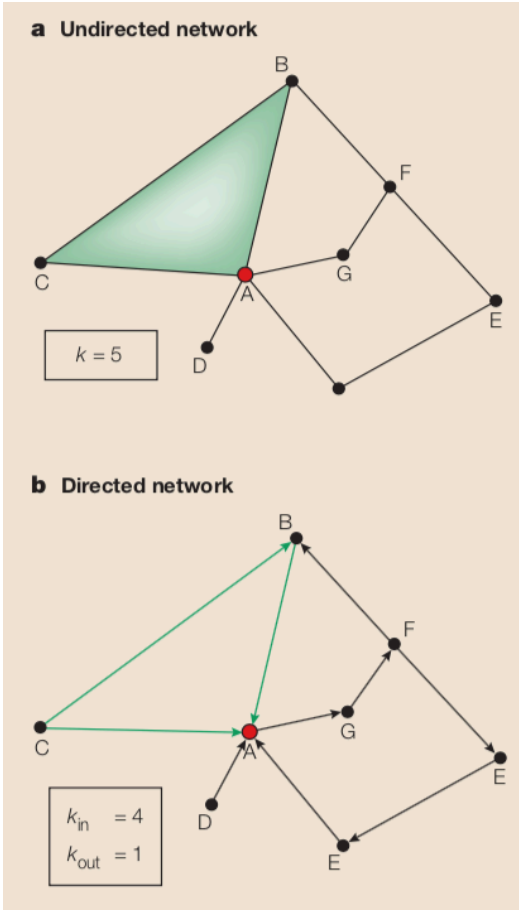
```
1. def get_successors(self, v):
2.     return list(self.graph[v])
3.
4. def get_predecessors(self, v):
5.     res = []
6.     for k in self.graph.keys():
7.         if v in self.graph[k]:
8.             res.append(k)
9.     return res
10.
11.def get_adjacents(self, v):
12.    suc = self.get_successors(v)
13.    pred = self.get_predecessors(v)
14.    res = pred
15.    for p in suc:
16.        if p not in res: res.append(p)
17.    return res
18.
```

Network measures

- **Degree**: how many links a node has. K_{in} = degree in and K_{out} = degree out. In an indirect network with N nodes and L links <K> = 2L / N (<> denotes average degree).

- **Shortest path**: path with the smallest number of links between selected nodes. Distances are measured as the length of the path.

- **Mean path length <*l*>**: average of all shortest path lengths between all pairs of nodes. Offers a measure of navigability in the network.



Figures adapted from: Network Biology: understanding the cell's functional organization
Albert-László Barabási & Zoltán N. Oltvai. Nature Reviews Genetics 2011.

Network measures

- **Degree distribution *P(k)***: gives the probability that a selected node has exactly k links. P(k) is obtained by counting how many nodes have k = 1,2,… links and dividing by the total number of nodes N. The degree distribution will allow to discriminate between different types of networks.

- **Scale free network**: the degree distribution approximates a power-law distribution where $P(k) \sim k^{-a}$
  - *a* determines the properties of the systems; the smaller the value of *a*, the more important role of the hubs in the network.
  - Hubs are highly connected nodes.
  - Networks with a power law degree distribution are “scale-free”!
  - Most networks on the cell are scale-free.

Network measures

- **Clustering coefficient**: indicates how connected are the the nodes on the network. If a node A is connected to B and B is connected to C, then it is highly probable that A is connected to C. The clustering coefficient of a node, gives the number of triangles that go through that node.

  $C_x = 2\, n_x / K \times (K - 1)$

  - n_x - number of links that connect the neighbours of x (node of triangles that pass through x)
  - K - degree of x.
  - K x (K -1) number of triangles that could pass through x if all neighbours are linked to each other.

- **Average clustering coefficient, <C>**, characterises the overall tendency of nodes to form clusters or groups.

- **C(k)**: average clustering coefficient of all nodes with k links. Many real networks have $C(k) \sim k^{-1}$ indicating a highly hierarchical structure of the network.

Network measures

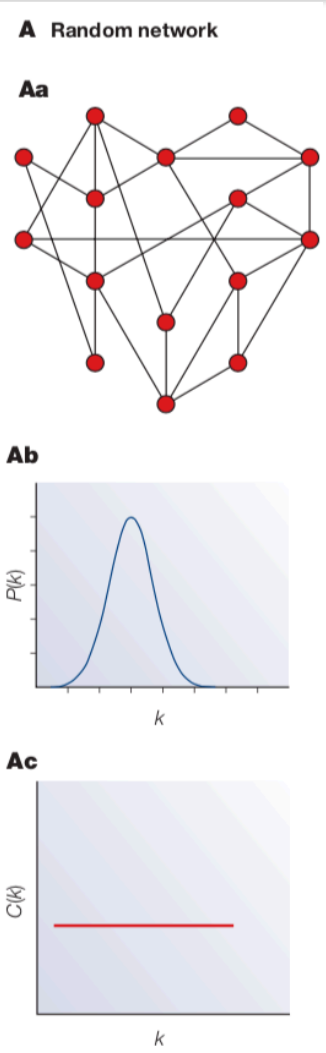
- Measures dependent on the number of nodes (N) and links (L):
  - average degree <k>
  - average path length <l>
  - average clustering coefficient <C>

- Independent of network size:
  - Degree distribution P(k)
  - Clustering coefficient C(k)

- These last two measures capture the features and characteristics of the network and therefore allow to compare and classify the various networks.

Random Networks

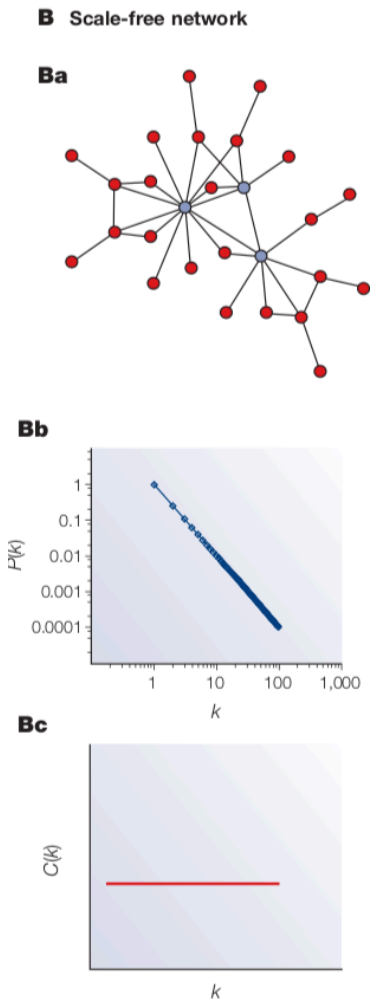
- Most nodes have approximately the same number of links (<k> is similar).
- P(k) decreases exponentially, which indicates that nodes that deviate from average are extremely rare.
- C(k) is independent of node degree.
- l ~ log N, indicating a small-world property.



Figures adapted from: Network Biology: understanding the cell's functional organization
Albert-László Barabási & Zoltán N. Oltvai. Nature Reviews Genetics 2011.

Scale-free Networks

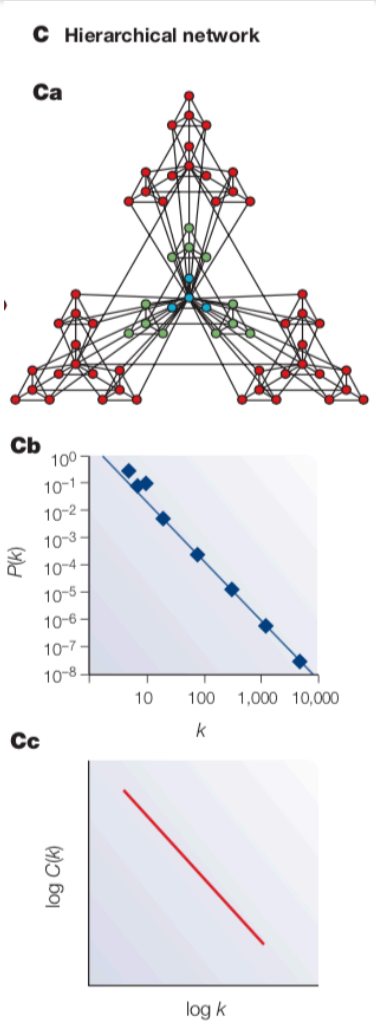
- P(k) follows a power law $P(k) \sim k^{-a}$, with *a* as degree exponent.
- The probability that a node is highly connected is more significant than in a random network.
- The network structure is often determined by a relatively small number of highly connected nodes (hubs).
- Many biological networks have $2 < a < 3$ and $\langle l \rangle \sim \log \log N$.
- C(k) is independent of node degree k.



Figures adapted from: Network Biology: understanding the cell's functional organization
Albert-László Barabási & Zoltán N. Oltvai. Nature Reviews Genetics 2011.

Hierarchical Networks

- Co-existence of modularity
- The starting point is a small number of four nodes highly linked.
- Communication between highly clustered models is maintained by hubs.
- Scale free topology + modular structure
- Clustering coefficient follows $C(k) \sim K^{-1}$ is the most important signature for this type of networks.



Figures adapted from: Network Biology: understanding the cell's functional organization
Albert-László Barabási & Zoltán N. Oltvai. Nature Reviews Genetics 2011.

MyGraph class

- Calculate the following degree functions:
  - **mean_degree** as the average of all node degrees
  - **prob_degree** as the frequency of the node degrees in the network

MyGraph class

- Calculate the following degree functions:
  - **mean_degree** as the average of all node degrees
  - **prob_degree** as the frequency of the node degrees in the network

```
1. def mean_degree(self, deg_type = "inout"):
2.     degs = self.all_degrees(deg_type)
3.     return sum(degs.values()) / float(len(degs))
4.
5. def prob_degree(self, deg_type = "inout"):
6.     degs = self.all_degrees(deg_type)
7.     res = {}
8.     for k in degs.keys():
9.         if degs[k] in res.keys():
10.            res[degs[k]] += 1
11.        else:
12.            res[degs[k]] = 1
13.    for k in res.keys():
14.        res[k] /= float(len(degs))
15.    return res
16.
```

Paths

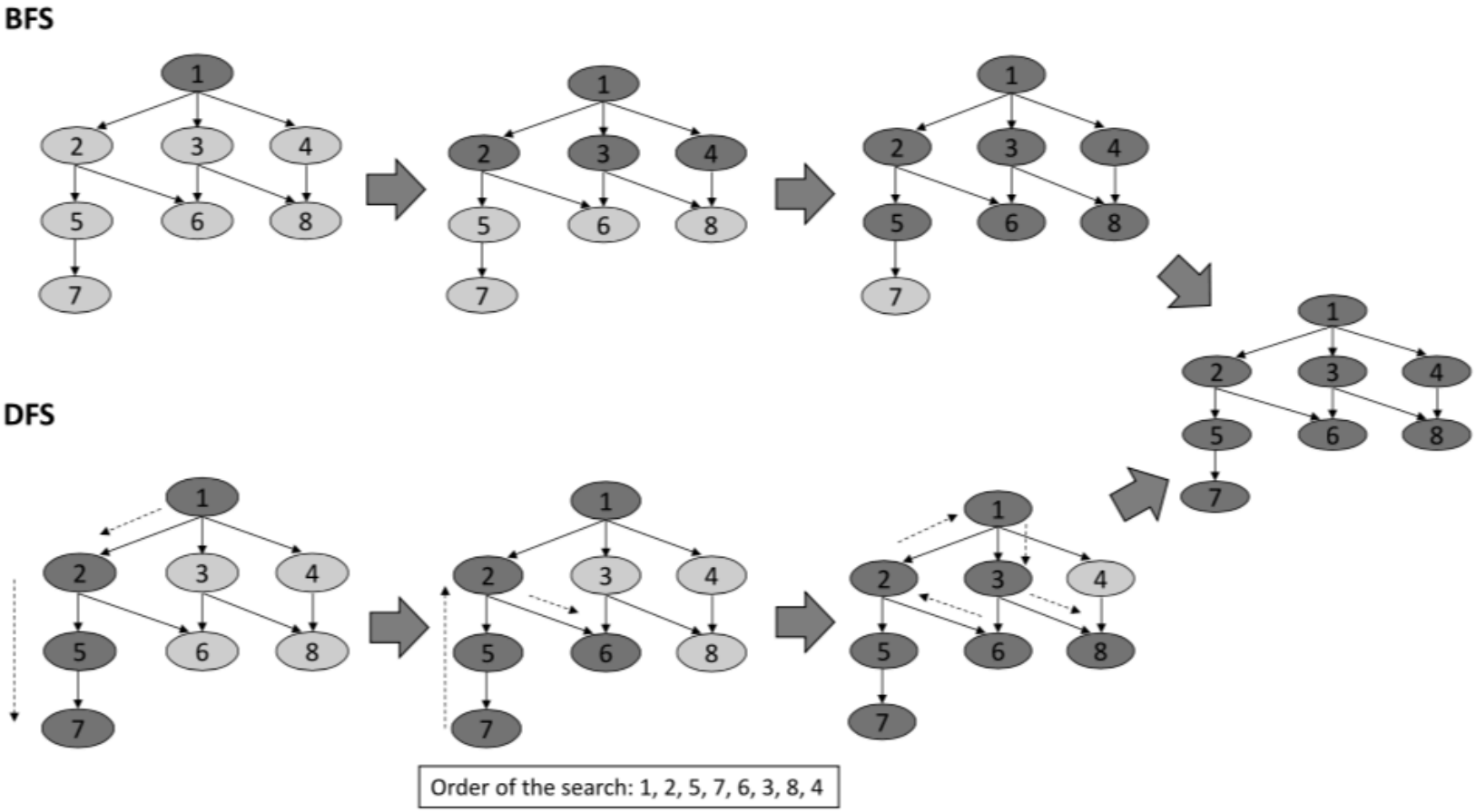
- **Path** is an ordered list of nodes, where consecutive nodes in the list are connected by an edge.
- Two nodes *u* and *v* are considered to be **connected** if there is a valid path between them (*u* is **reachable** from *v*).
- **Shortest path** is the path connecting the two nodes with the shortest length.
- **Distance** between nodes *u* and *v* corresponds to the number of edges in the shortest path.

Paths

- Algorithms that traverse the graph at a given source node and gather the visiting nodes.

- Two strategies to traverse the graph:
  - Breadth-First Search (BFS): starts with the source node, then visits all its successors, followed by their successors until all nodes are visited.
  - Depth-First Search (DFS): starts with the source node, explores its first successor, then its first sucessor until no further exploration is possible. Backtracks to explore further alternatives.

DFS and BFS



- From Bioinformatics Algorithms, Rocha & Ferreira

DFS and BFS


```
1. def reachable_bfs(self, v):
2.     l = [v]   # list of nodes to be handled
3.     res = []  # list of nodes to return the result
4.     while len(l) > 0:
5.         node = l.pop(0)  # implements a queue: LILO
6.         if node != v: res.append(node)
7.         for elem in self.graph[node]:
8.             if elem not in res and elem not in l and elem != node:
9.                 l.append(elem)
10.    return res
11.
12.def reachable_dfs(self, v):
13.    l = [v]
14.    res = []
15.    while len(l) > 0:
16.        node = l.pop(0) # implements a stack:
17.        if node != v: res.append(node)
18.        s = 0
19.        for elem in self.graph[node]:
20.            if elem not in res and elem not in l:
21.                l.insert(s, elem)
22.                s += 1
23.    return res
24.
```

Distance and shortest path

- Distance and shortest path can be derived from DFS and BFS traversal.
- BFS shows an interesting property with all nodes at distance n from the source appearing first then nodes at distance n+1.
- In distance function the working list keeps already visited nodes and their distances to source.

```
1. def distance(self, s, d):
2.     if s == d: return 0
3.     l = [(s,0)]
4.     visited = [s]
5.     while len(l) > 0:
6.         node, dist = l.pop(0)
7.         for elem in self.graph[node]:
8.             if elem == d: return dist + 1
9.             elif elem not in visited:
10.                print ((elem,dist+1))
11.                l.append((elem,dist+1))
12.                visited.append(elem)
13.    return None
14.
```

Distance and shortest path

- In shortest path the working list keeps visited nodes and the path to the source node. It will stop when destination is reached.

```
1. def shortest_path(self, s, d):
2.     if s == d: return 0
3.     l = [(s,[])]
4.     visited = [s]
5.     while len(l) > 0:
6.         node, preds = l.pop(0)
7.         for elem in self.graph[node]:
8.             if elem == d: return preds+[node,elem]
9.             elif elem not in visited:
10.                l.append((elem,preds+[node]))
11.                visited.append(elem)
12.    return None
13.
```

Network measures

- Not all nodes are equally significant. Identification of nodes that are most important is a central task in network analysis.

- Identification of hub nodes provides information on important properties of the network, like the robustness and resistance to failure. In biological networks a hub node in a gene-gene or gene-protein interaction network may reveal an important regulator, for instance a transcription factor that regulates many other genes.

- Centrality measures provides criteria to evaluate the importance of the node.

- The number of neighbours of a node, i.e. the node degree is one of the centrality measures - degree centrality.

Exercises

- Implement the following methods:
  - **clustering_coef** as clustering coefficient of a given node.
  - **all_clustering_coefs** returns a dictionary with all the clustering coefficient for all the nodes in the graph
  - **mean_clustering_coef** returns the mean value of all clustering coefficient for all the nodes in the graph
  - Create your own graph and test all the implemented and provided methods

- DOT is a graph description language. It is interpreted by several graph plotting programs including GraphViz. Check the syntax of the DOT and create a parser that given the example file uses the MyGraph class to create a graph. Run the different measures and obtain a list of features on the graph.