

Redes biológicas

Tipos de redes, construção, análise topológica

Redes biológicas

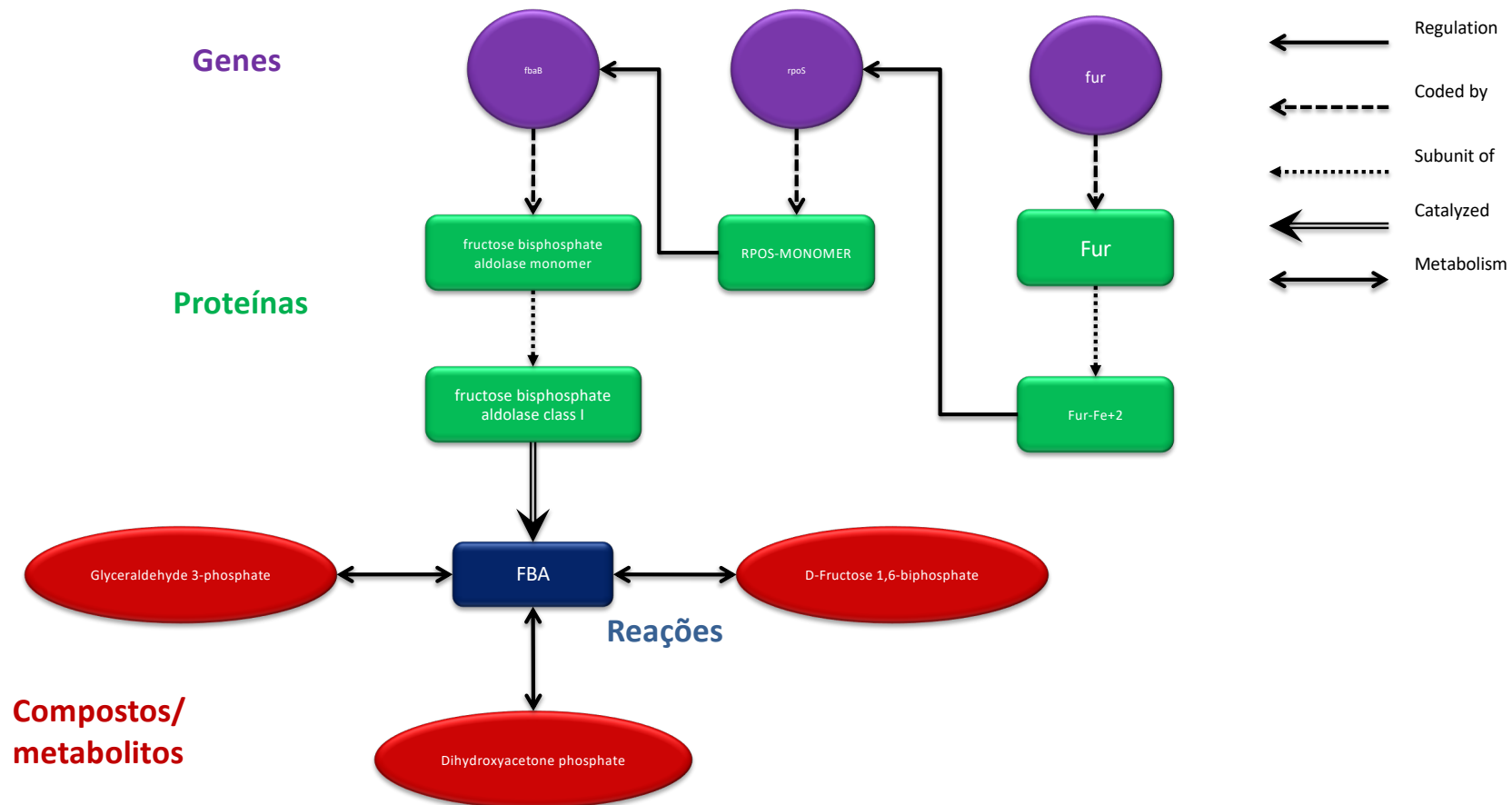
Modelos baseados em grafos têm sido usados em Bioinformática/ Biologia de Sistemas para representar, entre outras:

- **Redes genéticas** (regulatórias, de expressão)
- Redes **metabólicas** (reações e metabolitos)
- Redes de **transdução de sinal**

Grafos usados para representar a estrutura das redes biológicas:

- **Nós** representam **entidades** biológicas (genes, proteínas, metabolitos, etc)
- **Ligações** representam **relações / interações** (e.g. FTs regulam genes, genes codificam proteínas, etc)

Redes biológicas: exemplo



Redes metabólicas

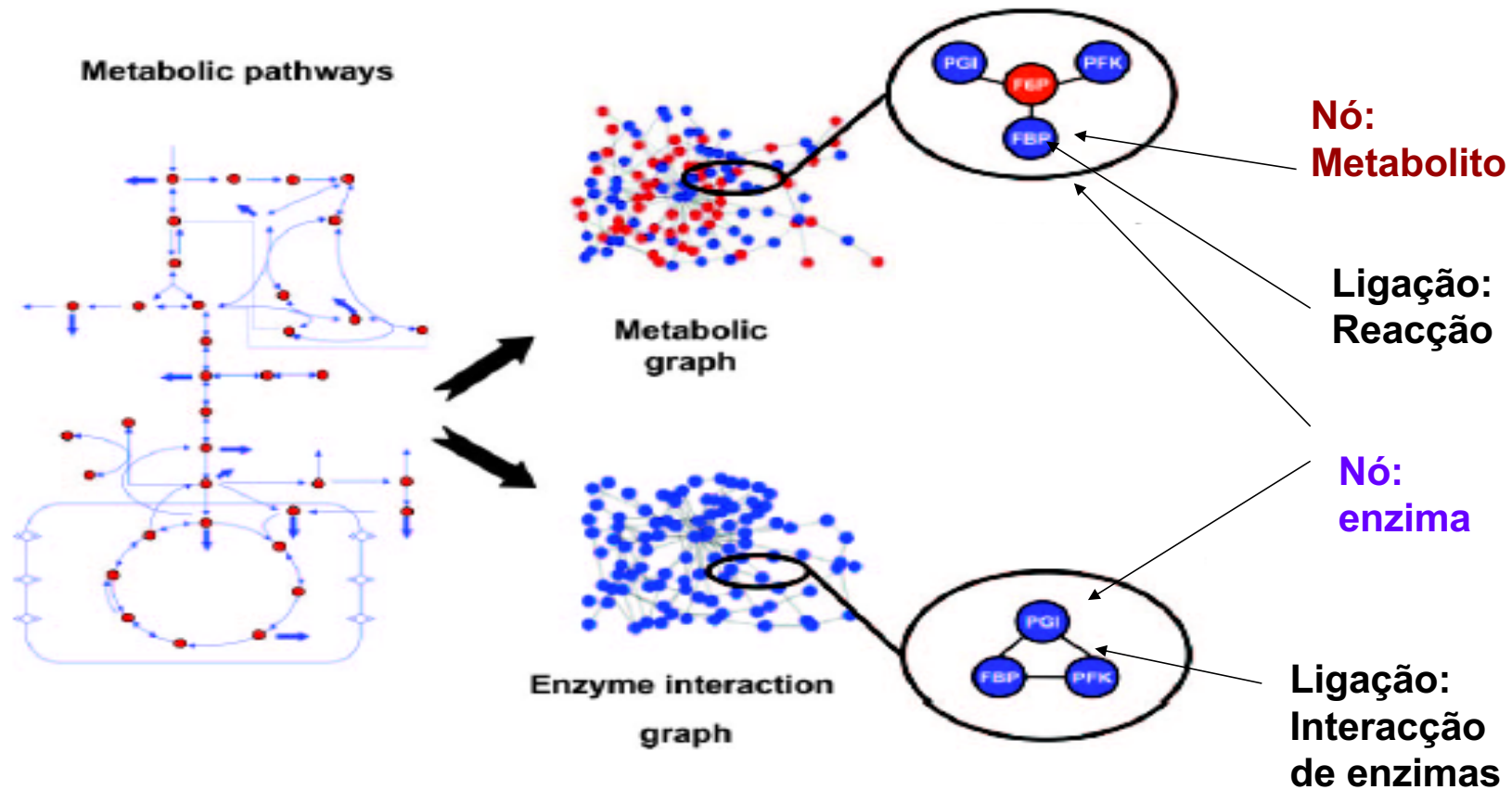
Redes metabólicas representam o metabolismo, i.e. o conjunto de reações químicas e os compostos envolvidos

Nós da rede representam reações (ou enzimas) e/ ou compostos (substratos e produtos das reações)

Vários tipos de rede possíveis:

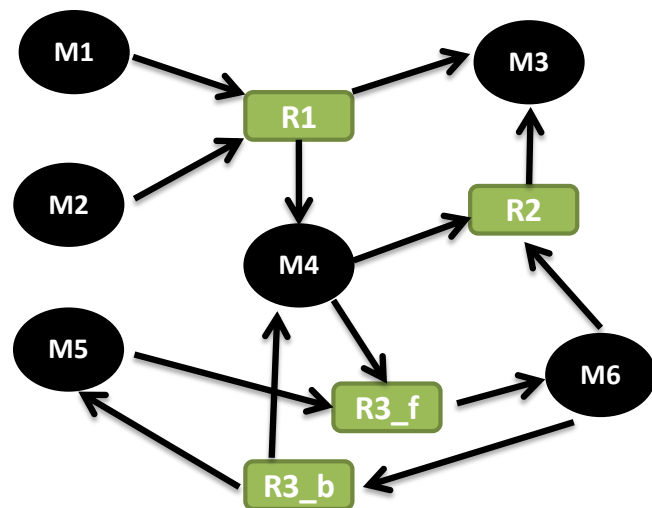
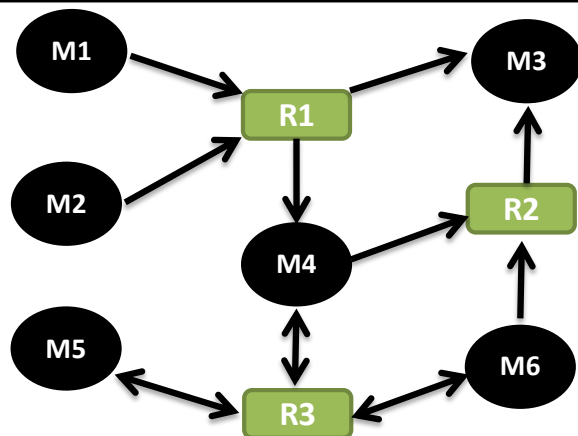
- Rede de **metabolitos**, onde nós são os compostos e reações são representadas pelos arcos do grafo
- Rede de **reações**, onde nós são reações e arcos representam ligações entre reações por metabolitos comuns
- Rede de **metabolitos e reações**, onde nós são compostos e reações, sendo os arcos indicativos da participação dos compostos em reações como substratos ou produtos

Redes metabólicas: exemplos



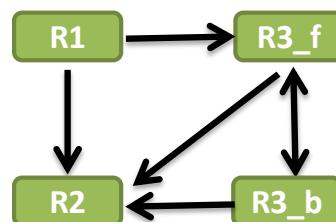
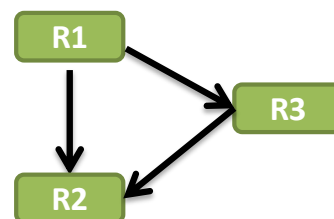
Patil and Nielsen (2005), PNAS, 102, 2685-9

Rede de metabolitos e reações

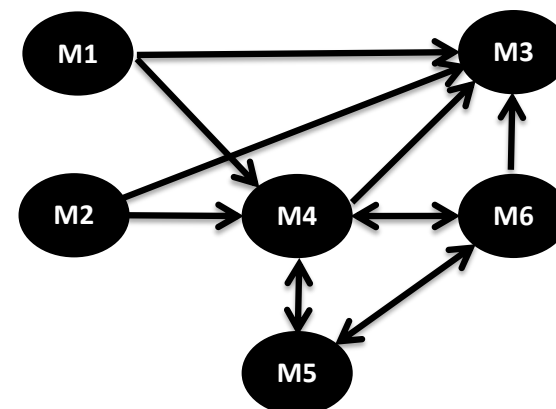


Redes metabólicas: exemplos

Rede de reações



Rede de metabolitos



Sistema metabólico

R1: $M1 + M2 \Rightarrow M3 + M4$
 R2: $M4 + M6 \Rightarrow M3$
 R3: $M4 + M5 \rightleftharpoons M6$



R1: $M1 + M2 \Rightarrow M3 + M4$
 R2: $M4 + M6 \Rightarrow M3$
 R3_f: $M4 + M5 \Rightarrow M6$
 R3_b: $M6 \Rightarrow M4 + M5$

Grafos bipartidos

Um grafo $G=(V,E)$ é dito **bipartido** quando:

- o seu conjunto de vértices V pode ser dividido em dois conjuntos $V1$ e $V2$, tais que a reunião de $V1$ e $V2$ é igual a V , e a sua interseção é vazia;
- O conjunto de arcos E só contém pares onde um dos vértices pertence a $V1$ e o outro a $V2$, i.e. só existem ligações entre elementos de cada um dos dois sub-conjuntos e não existem ligações entre elementos do mesmo sub-conjunto

As redes metabólicas reação-metabolito são grafos bipartidos onde $V1$ = conjunto de metabolitos e $V2$ = conjunto de reações

Implementando redes metabólicas

Com base no código genérico para grafos orientados (classe **MyGraph**) vamos criar uma classe para representar e analisar os vários tipos de rede metabólica

Esta classe será uma **sub-classe** da classe **MyGraph**, **herdando** assim todos os seus atributos e métodos

- Classe **MetabolicNetwork** irá representar redes metabólicas, estendendo classe **MyGraph**
- Atributos da classe:
 - **graph** – dicionário herdado da classe **MyGraph**
 - **net_type** – tipo de rede: “metabolite-reaction”, “reaction”, “metabolite”
 - **node_types** – guarda dicionário indicando listas de nós de cada tipo (no caso de ser uma rede “metabolite-reaction” – grafo bipartido)
 - **split_rev** – indica se se consideram as reações reversíveis como duas reações distintas, uma para cada direção (True) ou não (False)

Implementando redes metabólicas

```
from MyGraph import MyGraph
```

```
class MetabolicNetwork (MyGraph):
```

```
    def __init__(self, network_type = "metabolite-reaction", split_rev = False):
```

```
        MyGraph.__init__(self, {})
```

```
        self.net_type = network_type
```

```
        self.node_types = {}
```

```
        if network_type == "metabolite-reaction":
```

```
            self.node_types["metabolite"] = []
```

```
            self.node_types["reaction"] = []
```

```
        self.split_rev = split_rev
```

Sub-classe da
classe
MyGraph



Implementando redes metabólicas

```
def add_vertex_type (self, v, nodetype):  
    self.add_vertex(v)  
    self.node_types[nodetype].append(v)  
  
def get_nodes_type (self, node_type):  
    if node_type in self.node_types:  
        return self.node_types[node_type]  
    else: return None
```

Novas funções para
adicionar nós e arcos

Implementando redes metabólicas

```
def test1():  
    m = MetabolicNetwork("metabolite-reaction")  
    m.add_vertex_type("R1","reaction")  
    (...)  
    m.add_vertex_type("M1","metabolite")  
    (...)  
    m.add_edge("M1","R1")  
    (...)  
    m.add_edge("M6","R3")  
  
    m.print_graph()  
    print("Reactions: ", m.get_nodes_type("reaction") )  
    print("Metabolites: ", m.get_nodes_type("metabolite") )
```

Implementando redes metabólicas

Para criar redes de sistemas reais de grande dimensão é conveniente poder ler a informação de ficheiros

Vamos definir uma função que lê um ficheiro onde cada reação é definida numa linha

Exemplo:

Ficheiro de texto

“example-net.txt”

R1: M1 + M2 => M3 + M4

R2: M4 + M6 => M3

R3: M4 + M5 <=> M6

Implementando redes metabólicas

```
def load_from_file(self, filename):  
    rf = open(filename)  
    gmr = MetabolicNetwork("metabolite-reaction")  
    for line in rf:  
        (...)  
  
    if self.net_type == "metabolite-reaction":  
        self.graph = gmr.graph  
        self.node_types = gmr.node_types  
    elif self.net_type == "metabolite-metabolite":  
        self.convert_metabolite_net(gmr)  
    elif self.net_type == "reaction-reaction":  
        self.convert_reaction_graph(gmr)  
    else: self.graph = {}
```

Primeiro criamos uma rede "metabolite-reaction" (grafo bipartido)

Se quisermos outro tipo de rede convertemos da anterior

Implementando redes metabólicas

```
def convert_metabolite_net(self, gmr):  
    for m in gmr.node_types["metabolite"]:  
        ...
```

```
def convert_reaction_graph(self, gmr):  
    for r in gmr.node_types["reaction"]:  
        ...
```

Implementando redes metabólicas

```
def convert_metabolite_net(self, gmr):  
    for m in gmr.node_types["metabolite"]:  
        self.add_vertex(m)  
        sucs = gmr.get_successors(m)  
        for s in sucs:  
            sucs_r = gmr.get_successors(s)  
            for s2 in sucs_r:  
                if m != s2:  
                    self.add_edge(m, s2)
```

```
def convert_reaction_graph(self, gmr):  
    for r in gmr.node_types["reaction"]:  
        self.add_vertex(r)  
        sucs = gmr.get_successors(r)  
        for s in sucs:  
            sucs_r = gmr.get_successors(s)  
            for s2 in sucs_r:  
                if r != s2: self.add_edge(r, s2)
```

Implementando redes metabólicas

```
def test2():  
    print("metabolite-reaction network:")  
    mrn = MetabolicNetwork("metabolite-reaction")  
    mrn.load_from_file("example-net.txt")  
    mrn.print_graph()  
    print("Reactions: ", mrn.get_nodes_type("reaction") )  
    print("Metabolites: ", mrn.get_nodes_type("metabolite") )  
    print()
```

Vamos testar o nosso código com o ficheiro exemplo
Verifique se o grafo imprimido corresponde ao esperado

Como fazer para criar os outros dois tipos de rede ? :

- Só Metabolitos
- Só Reações

Implementando redes metabólicas

```
print("metabolite-metabolite network:")
mmn = MetabolicNetwork("metabolite-metabolite")
mmn.load_from_file("example-net.txt")
mmn.print_graph()
print()

print("reaction-reaction network:")
rrn = MetabolicNetwork("reaction-reaction")
rrn.load_from_file("example-net.txt")
rrn.print_graph()
print()
```

De novo, verifique se os grafos correspondem ao esperado ...

Teste o mesmo código para as redes com reações com a flag “split_rev” a True

Redes metabólicas: exemplo

Para testar o código anterior com um exemplo maior, use o ficheiro: “*ecoli.txt*”

Estes foram criados a partir de um modelo de escala genómica de *Escherichia coli* (iJR904)

O ficheiro tem a mesma estrutura do anterior pelo que o código deve poder ser usado sem grandes alterações

Verifique o tamanho dos grafos gerados

```
def size(self):  
    return len(self.getNodes()), len(self.getEdges())
```

Função adicionada a classe MyGraph

Análise topológica de redes: graus e distribuição de graus

Grau médio $\langle k \rangle$ - média do grau calculada sobre todos os nós (pode ser calculado apenas para graus de entrada/ saída em grafos orientados)

Distribuição do grau $P(k)$: probabilidade que um nó tenha grau k . $P(k)$ é independente do tamanho da rede

Redes “**Scale-free**”

- Distribuição dos graus aproxima a *power law* $P(k) \sim k^{-\gamma}$ ($2 < \gamma < 3$)
- Isto implica que há poucos nós com muitas ligações e muitos nós com poucas ligações

Análise topológica de redes: caminhos mais curtos

Distância: quantas ligações temos que passar para viajar entre dois nós, considerando o caminho mais curto

Comprimento médio dos caminhos mais curtos $\langle L \rangle$ - média dos comprimentos dos caminhos mais curtos entre todos os pares de nós

Redes “Small world”

- Valor de $\langle L \rangle$ é pequeno (quando comparado com redes geradas aleatoriamente)
- Isto significa que nós estarão mais perto do que o que seria expectável

Coeficiente de clustering

Em algumas aplicações, é importante identificar até que ponto os nós de um grafo tendem a agrupar-se , i.e. a criar componentes do grafo com muitas ligações entre si

Para medir até que ponto cada nó está inserido num grupo coeso, é definido o **coeficiente de clustering**, que se define para cada nó como:

nº de arcos existentes entre vizinhos do nó /

nº total de arcos que poderiam existir entre vizinhos do nó

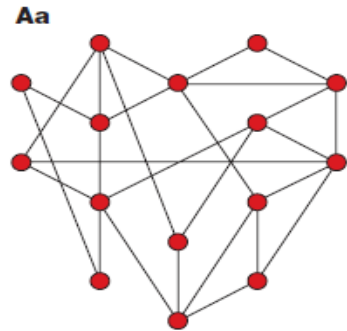
Análise topológica de redes: clustering

Média do coeficiente de clustering $\langle C \rangle$ - média sobre todos os nós

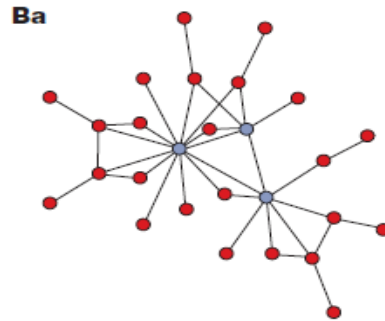
$C(k)$ – Média dos coeficientes considerando nós de grau k . $C(k)$ não depende do tamanho da rede

Análise topológica de redes teóricas

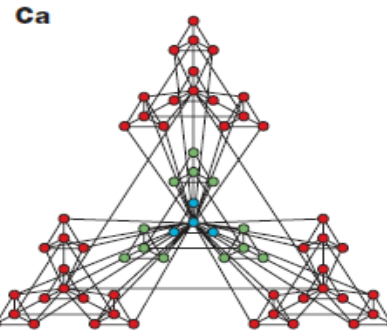
A Random network



B Scale-free network

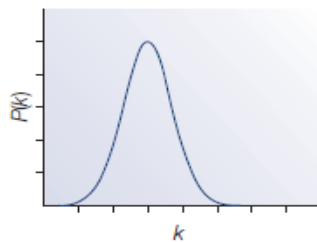


C Hierarchical network

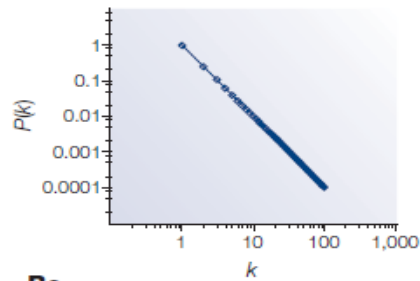


Classe da rede

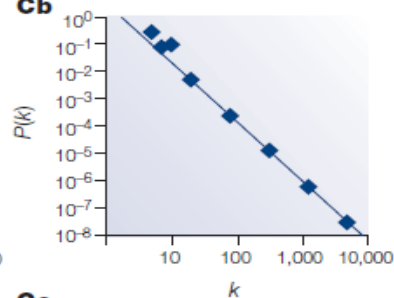
Ab



Bb



Cb

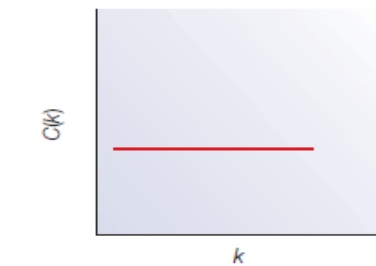


$P(k)$

Ac



Bc



Cc



$C(k)$

Barabasi et al, 2004

Implementação de graus

```
def all_degrees (self, deg_type = "inout"):
    degs = {}
    for v in self.graph.keys():
        if deg_type == "out" or deg_type == "inout":
            degs[v] = len(self.graph[v])
        else: degs[v] = 0
    if deg_type == "in" or deg_type == "inout":
        for v in self.graph.keys():
            for d in self.graph[v]:
                if deg_type == "in" or v not in self.graph[d]:
                    degs[d] = degs[d] + 1
    return degs
```

Cálculo de graus de entrada e saída (ou ambos) para todos os nós da rede

Adicionar à classe *MyGraph*

Implementação de graus

```
def mean_degree (self, deg_type = "inout"):
    degs = self.all_degrees(deg_type)
    return sum(degs.values()) / float(len(degs))
```

```
def prob_degree (self, deg_type = "inout"):
    degs = self.all_degrees(deg_type)
    res = {}
    for k in degs.keys():
        if degs[k] in res.keys():
            res[degs[k]] += 1
        else:
            res[degs[k]] = 1
    for k in res.keys():
        res[k] /= float(len(degs))
    return res
```

```
print (mrn.mean_degree("out"))
d = mrn.prob_degree("out")
for x in sorted(d.keys()):
    print (x, "\t", d[x])
```

Aplique as funções anteriores à rede criada para a *E. coli*. O que conclui ?

Implementação de distância média

```
def mean_distances(self):  
    tot = 0  
    num_reachable = 0  
    for k in self.graph.keys():  
        distsk = self.reachable_with_dist(k)  
        for _, dist in distsk:  
            tot += dist  
        num_reachable += len(distsk)  
    meandist = float(tot) / num_reachable  
    n = len(self.getNodes())  
    return meandist, float(num_reachable)/((n-1)*n)
```

Adição ao código anterior (*MyGraph*)
Ignora nós não atingíveis

distância média

proporção de nós atingíveis

Aplique a função anterior à rede criada para a *E. coli*. O que conclui ?

```
def clustering_coef(self, v):
    adjs = self.get_adjacents(v)
    if len(adjs) <= 1: return 0.0
    ligs = 0
    for i in adjs:
        for j in adjs:
            if i != j:
                if j in self.graph[i] or i in self.graph[j]:
                    ligs = ligs + 1
    return float(ligs)/(len(adjs)*(len(adjs)-1))
```

Implementando grafos: clustering

```
gr = MyGraph( {1:[2], 2:[3], 3:[2,4], 4:[2]} )
print (gr.clustering_coef(1))
print (gr.clustering_coef(2))
```

```
gr2 = MyGraph( {1:[2,3], 2:[4], 3:[5], 4:[], 5:[]} )
print (gr2.clustering_coef(1))
```

```
gr3 = MyGraph( {1:[2,3], 2:[1,3], 3:[1,2]} )
print (gr3.clustering_coef(1))
```

Implementando grafos: clustering

```
def all_clustering_coefs (self):  
    ccs = {}  
    for k in self.graph.keys():  
        ccs[k] = self.clustering_coef(k)  
    return ccs  
  
def mean_clustering_coef(self):  
    ccs = self.all_clustering_coefs()  
    return sum(ccs.values()) / float(len(ccs))
```

Calcula todos os
coeficientes

Média global dos
coeficientes na rede

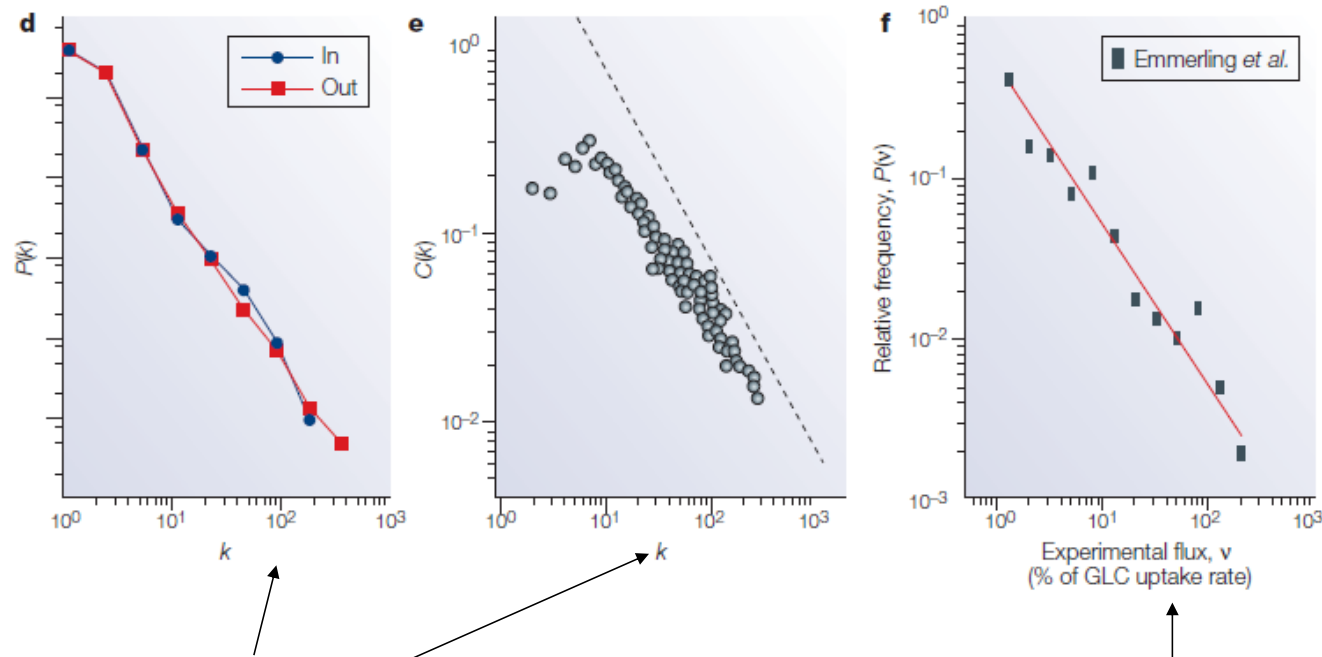
Implementando grafos: clustering

Calcula valores
para $C(k)$ para
todos os nós

```
def mean_clustering_perdegree (self, deg_type = "inout"):
    degs = self.all_degrees(deg_type)
    ccs = self.all_clustering_coefs()
    degs_k = {}
    for k in degs.keys():
        if degs[k] in degs_k.keys():
            degs_k[degs[k]].append(k)
        else: degs_k[degs[k]] = [k]
    ck = {}
    for k in degs_k.keys():
        tot = 0
        for v in degs_k[k]: tot += ccs[v]
        ck[k] = float(tot) / len(degs_k[k])
    return ck
```

Aplique à rede criada para a *E. coli* e analise os resultados

Análise topológica de redes metabólicas



$P(k)$ e $C(k)$ calculados para redes **metabólicas** (média de 43 organismos)

Distribuição de Fluxos de reações - *E.coli*

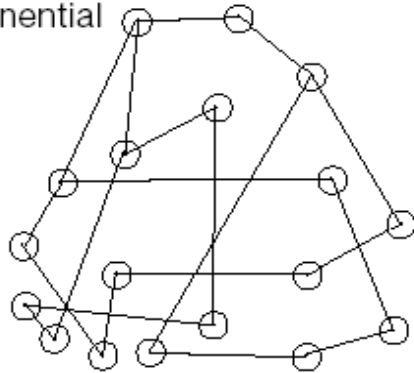
Barabasi et al, 2004

Análise topológica de redes metabólicas

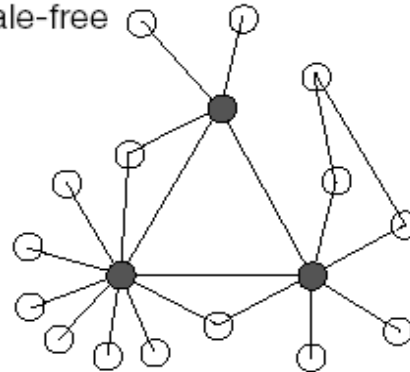
Análise das propriedades **topológicas** das redes metabólicas conduzem a conclusões importantes:

- Redes metabólicas são redes **scale-free**, com poucos nós altamente conectados (hubs) e uma maioria de nós pouco conectados.
- Redes metabólicas exibem características **small world**, uma vez que 2 nós estão sempre relativamente próximos (baixo $\langle L \rangle$)
- Redes metabólicas são **hierárquicas**

Exponential



Scale-free



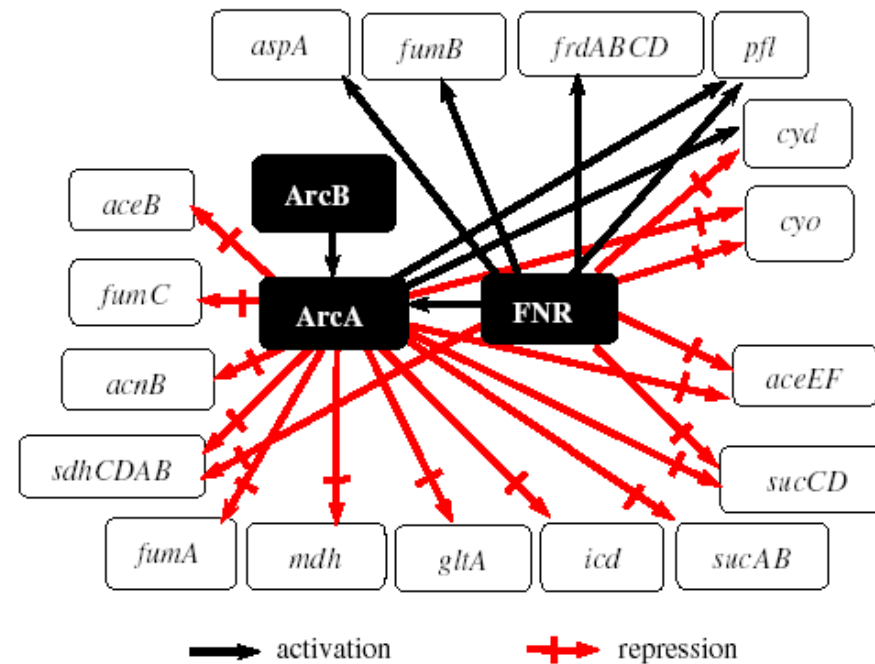
Jeong et al (2000),
Nature, 407, 651-4

Redes regulatórias

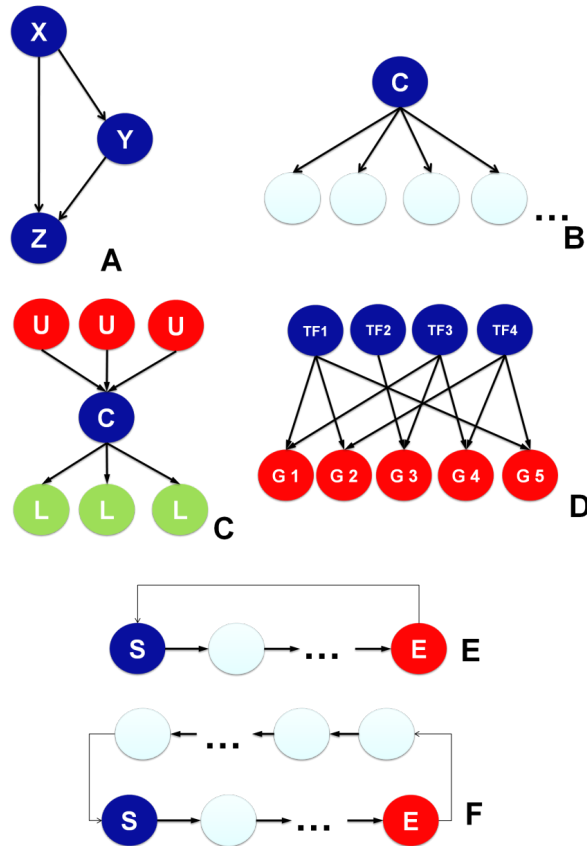
Redes regulatórias representam tipicamente fenómenos de regulação:

- Nós tipicamente são **genes** e/ ou **proteínas** regulatórias (e.g. fatores de transcrição)
- Arcos representam interações entre genes e proteínas em fenómenos de regulação (e.g. ativação ou inibição de um gene por um FT)

Redes regulatórias: exemplo



Motifs em redes biológicas: exemplos



A: Feed Forward Loop
B: Single Input Module
C: Bowtie
D: Dense Overlap Regulon
E-F: Cyclic patterns (Extended Feedback Loop and Double Extended Feedback Loop)

Motifs são tipicamente definidos como padrões sobre-representados, i.e. que aparecem com mais frequência do que seria esperado num grafo gerado “aleatoriamente”

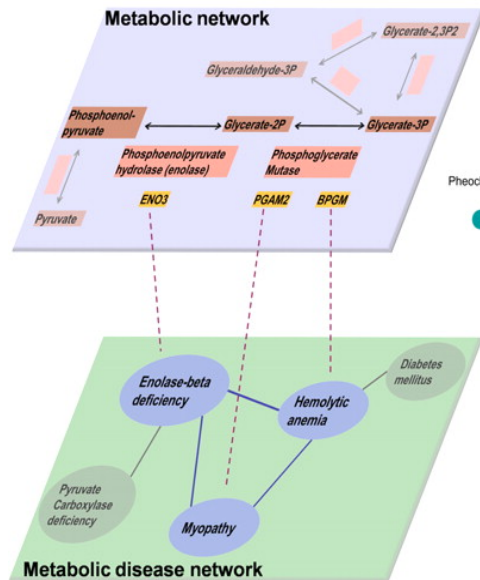
Difícil definir qual a melhor forma de gerar redes de forma “aleatória” ...

Aplicações: “network medicine”

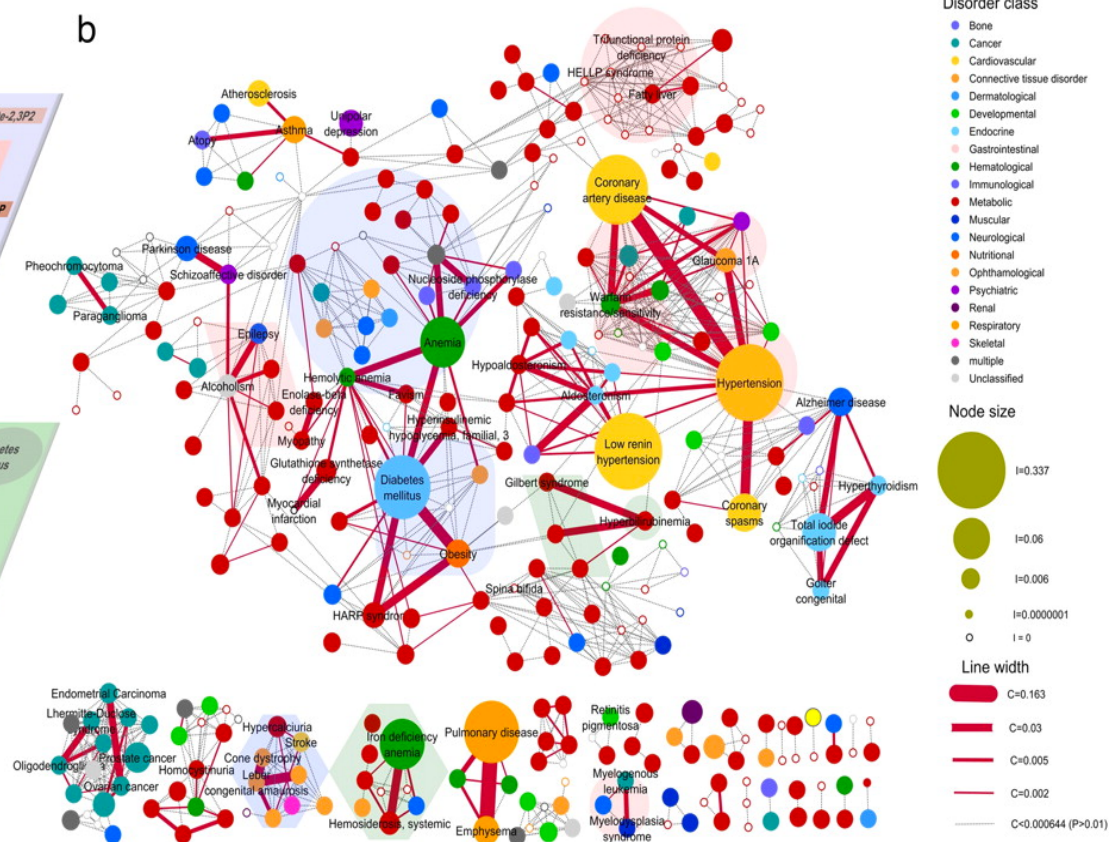
- Um fenótipo de uma doença raramente é consequência de uma anormalidade única, mas resulta normalmente de vários processos que interagem numa **rede complexa**
- Abordagens baseadas na análise de redes podem ter um potencial enorme em termos de **aplicações clínicas**
- Como exemplo, em redes metabólicas, um problema que afecte o fluxo de uma reação irá afectar o fluxo de todas as reações que se seguem nas vias metabólicas onde esteja integrada

Exemplo de redes “clínicas”

a



b



Duas doenças estão ligadas se as enzimas associadas com elas catalisam reações adjacentes (num grafo de reações)

Lee et al, PNAS, 2008

Leitura adicional

NETWORK BIOLOGY: UNDERSTANDING THE CELL'S FUNCTIONAL ORGANIZATION

Albert-László Barabási & Zoltán N. Oltvai‡*

A key aim of postgenomic biomedical research is to systematically catalogue all molecules and their interactions within a living cell. There is a clear need to understand how these molecules and the interactions between them determine the function of this enormously complex machinery, both in isolation and when surrounded by other cells. Rapid advances in network biology indicate that cellular networks are governed by universal laws and offer a new conceptual framework that could potentially revolutionize our view of biology and disease pathologies in the twenty-first century.

Network medicine: a network-based approach to human disease

*Albert-László Barabási**§, Natali Gulbahce**|| and Joseph Loscalzo§*

Abstract | Given the functional interdependencies between the molecular components in a human cell, a disease is rarely a consequence of an abnormality in a single gene, but reflects the perturbations of the complex intracellular and intercellular network that links tissue and organ systems. The emerging tools of network medicine offer a platform to explore systematically not only the molecular complexity of a particular disease, leading to the identification of disease modules and pathways, but also the molecular relationships among apparently distinct (patho)phenotypes. Advances in this direction are essential for identifying new disease genes, for uncovering the biological significance of disease-associated mutations identified by genome-wide association studies and full-genome sequencing, and for identifying drug targets and biomarkers for complex diseases.

Importância dos nós da rede

Existem várias formas de calcular métricas da importância (também designada por centralidade) dos nós das redes representadas por grafos, permitindo identificar os chamados *hubs*

- Baseadas nos graus dos nós (os mais elevados)
- Baseadas na proporção de caminhos mais curtos entre todos os nós que passam pelo nó (*betweenness centrality*)
- Baseadas nos nós que estão mais próximos dos restantes (*closeness centrality*)

$$CC(v) = \frac{N - 1}{\sum_{x \in V} d(v, x)}$$

Exercício (avaliação contínua)

Implementar as três métricas de centralidade atrás referidas para cada nó, bem como funções que permitam aplicar estas métricas a toda a rede e identificar os nós com valores mais elevados

Aplicar as funções anteriores à rede da *E. coli* de metabolitos e verificar quais os que têm valores mais elevados em cada caso

Avaliação do potencial metabólico

Representação da rede na forma de grafo bipartido (“metabolite”-“reaction”) permite analisar o potencial metabólico da rede, i.e. que metabolitos podem ser produzidos a partir de um conjunto de metabolitos iniciais

Exercício opcional: escrever funções que permitam, dada uma rede MR com as reações reversíveis desdobradas:

- Determinar todas as reações ativas dado uma lista de metabolitos existentes (aquelas em que todos os substratos estão na lista)
- Determinar metabolitos que podem ser produzidos dada uma lista de reações ativas (os produtos destas reações)
- Calcular todos os metabolitos “finais” que poderão ser produzidos, dada uma lista de metabolitos iniciais, aplicando iterativamente as funções anteriores