Algoritmos Avançados de Bioinformática

Trabalho Prático 2

Implementação de Redes Metabólicas usando NetworkX

Relatório de Desenvolvimento

Grupo 6

Eduardo Cunha A71940

João Dias PG37982

Junho 2019

Biblioteca e suas funcionalidades

Para este trabalho a biblioteca de grafos adoptada foi a NetworkX. Há medida que a fomos estudando, percebemos que já tem implementada uma quantidade de funções de implementação e análise de grafos bastante considerável, levando a que este trabalho fosse uma adaptação desta biblioteca para a implementação de Redes Metabólicas.

Assim, partindo da classe *Metabolic Network* implementada nas aulas, criou-se uma nova classe com o mesmo nome, cujo construtor inicial consiste em:

```
class MetabolicNetwork ():

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

Assim a nova classe tem uma variável *DiGraph*, ou seja, um grafo orientado, que permite a correta representação de redes metabólicas, mantendo-se o resto do construtor semelhante ao definido nas aulas.

Todo o código está documentado com a explicação de todas as funções no ficheiro *Python* implementado.

Leitura de Ficheiros

Para a leitura dos ficheiros de entrada adaptamos a função *load_from_file* das aulas para ficheiros de texto. No entanto, o *NetworkX* tem funções implementadas que permitem a leitura de diversos formatos como *xml* e *gml*, sendo os últimos mais comuns na leitura de ficheiros de redes metabólicas, já que estes podem ser extraídos do KEGG.

Apesar de esta funcionalidade não ter sido implementada na sua totalidade, o código abaixo demonstra diferentes funções do *NetworkX* que permitem ler diferentes tipos de ficheiros, úteis para este tipo de redes:

```
def read_from_file(filename):
    term = filename.split('.')
    if term == 'gexf':
        edge_node = nx.read_gexf(filename)
    elif term == 'gml':
        edge_nodes = nx.read_gml(filename)
    elif term == 'gpickle':
        edge_nodes = nx.read_gpickle(filename)
    elif term == 'graphml':
        edge_nodes = nx.read_graphml(filename)
    elif term == 'leda':
        edge_nodes = nx.read_leda(filename)
    elif term == 'yaml':
        edge_nodes = nx.read_yaml(filename)
    elif term == 'sparsegraph6':
        edge_nodes = nx.read_sparse6(filename)
    elif term == 'pajek':
        edge_nodes = nx.read_pajek(filename)
```

Representação dos grafos

Com o auxílio do package do *Python*, *matplotlib.pyplot*, foi implementada a demonstração das redes metabólicas sob a forma de grafo. Esta representação distingue com cores diferentes as reações dos metabolitos (função *print_graph(self)*), e ainda a distinção do caminho mais curto entre dois dados nodos (função *print_graph_sp(self,source,dest)*).
De notar que para representar alguma rede metabólica com um tamanho mais considerável, como a do ficheiro *ecoli.txt*, é necessário alterar o parâmetro *pos* destas funções, aumentando consideravelmente o segundo argumento.

Ficheiro example-net.txt

Após a análise do ficheiro *example-net.txt*, este é representado da seguinte forma:

print_graph

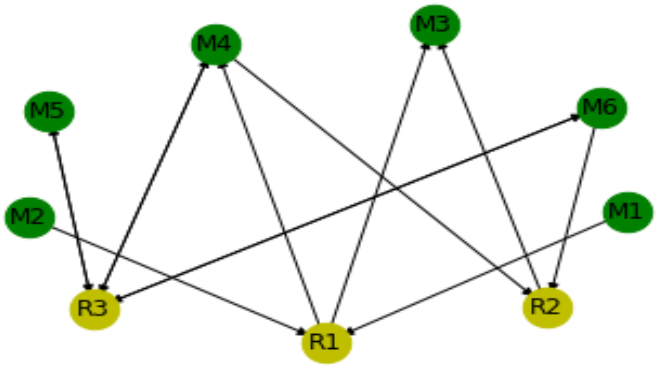

Figura 1: Representação do ficheiro *example-net.txt*

print_graph_sp("M5","M3")

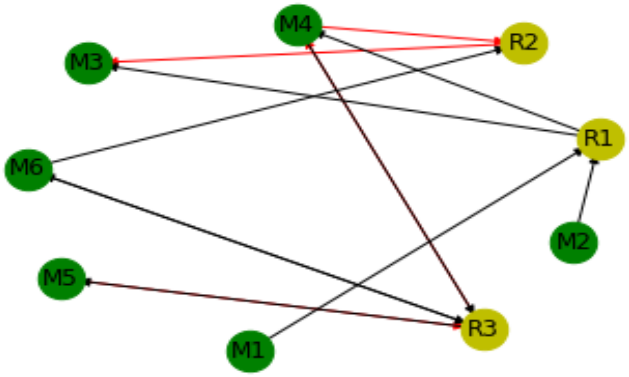

Figura 2: Representação do caminho mais curto *example-net.txt*

De notar que o caminho mais curto entre os nodos M5 e M3 está representado a vermelho, sendo que, quando existe caminho no sentido contrário que não faz parte do caminho mais curto, a visualização deste não é a melhor, sendo no entanto visível.

Introdução da Rede Metabólica pelo Python

Usando um dos casos de teste das aulas, decidimos criar uma rede metabólica pelo código, de modo a verificar a sua funcionalidade:

```
def test1(m):
m.add_vertex_type("R1","reaction")
m.add_vertex_type("R2","reaction")
m.add_vertex_type("R3","reaction")
m.add_vertex_type("M1","metabolite")
m.add_vertex_type("M2","metabolite")
m.add_vertex_type("M3","metabolite")
m.add_vertex_type("M4","metabolite")
m.add_vertex_type("M5","metabolite")
m.add_vertex_type("M6","metabolite")
m.add_edge("M1","R1")
m.add_edge("M2","R1")
m.add_edge("R1","M2")
m.add_edge("R1","M4")
m.add_edge("M4","R2")
m.add_edge("M6","R2")
m.add_edge("R2","M3")
m.add_edge("M4","R3")
m.add_edge("M5","R3")
m.add_edge("R3","M6")
m.add_edge("R3","M4")
m.add_edge("R3","M5")
m.add_edge("M6","R3")

print("Reactions: ", m.get_nodes_type("reaction") )
print("Metabolites: ", m.get_nodes_type("metabolite") )

return m
```

print_graph_sp("M5","M3")

```
Reactions:  ['R1', 'R2', 'R3']
Metabolites:  ['M1', 'M2', 'M3', 'M4', 'M5', 'M6']
```

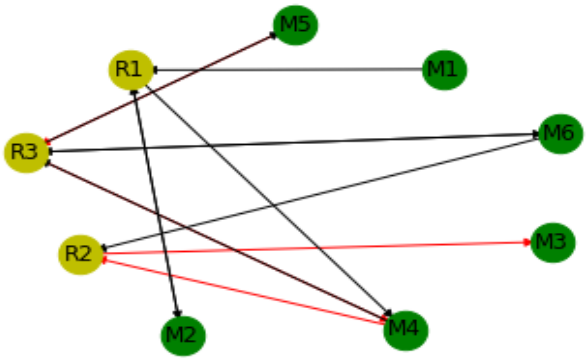

Figura 3: Representação do caminho mais curto da rede criada