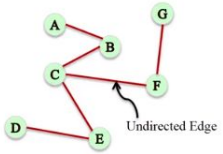
Tipos de rede

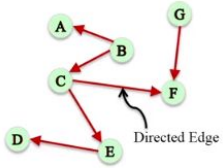
Undirected

import networkx as nx
G = nx.Graph()
G.add_edge('C', 'F')



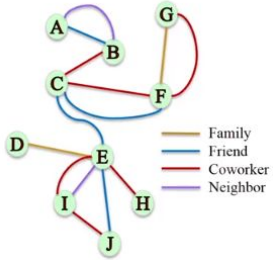
Directed

import networkx as nx
G = nx.DiGraph()
G.add_edge('C', 'F')

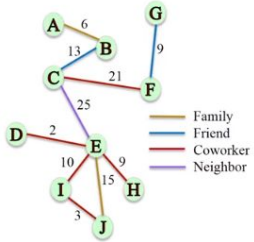


Multigraph

import networkx as nx
G = nx.MultiGraph()
G.add_edge('A', 'B', relation='friend')
G.add_edge('A', 'B', relation='neighbor')



Atributos de edges



Weighted
G.add_edge ('A', 'B', weighted = 10)

Sign
G.add_edge('A', 'B', sign = '+')

Relation
G.add_edge('A', 'B', relation = friend)

Acessando atributos

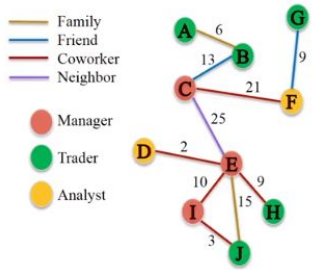
Lista edges e seus atributos
G.edges(data=True)

Lista edges e atributos específicos
G.edges(data='relation')

Acessar atributos de uma determinada edge
G.edge['A']['B']

Acessar atributo específico de uma determinada edge
G.edge['A']['B']['weight']

Atributos de nodes



Role
G.add_node('A', role = 'trader')

Acessando atributos

Lista nodes
G.nodes()

Lista nodes e seus atributos
G.edges(data=True)

Acessar atributo específico de um determinado node
G.node['A']['role']

Removendo edges e nodes
G.remove_edge('A', 'B')
G.remove_node('A')

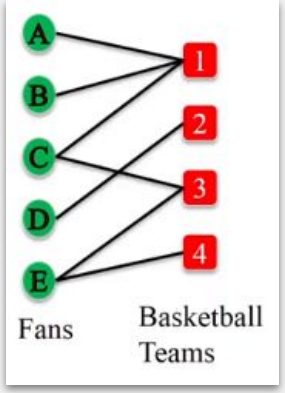
hacking analytics

https://hackinganalytics.com/
Fonte: Applied Social Network Analysis in Python - University of Michigan

Bipartite graphs

Neste tipo de rede os nodes podem ser divididos em dois grupos, sendo que cada conexão deve ser feita entre um node de cada grupo

Rede de exemplo:



Criando e acessando atributos de uma rede bipartite

Cria uma rede bipartite

```
from  networkx.algorithms import bipartite
B.nx.Graph()
B.add_nodes_from(['A','B','C','D','E'], bipartite=0)
B.add_nodes_from([1,2,3,4], bipartite=1)
B.add_edges_from([('A',1), ('B',1), ('C',1), ('C',3), ('D',2), ('E',3), ('E',4)])
```

Verifica se a rede é bipartite

```
bipartite.is_bipartite(B)
```

Verifica se um conjunto de nodes é uma partição

```
X = set([1,2,3,4])
bipartite.is_bipartite_node_set (B, X)
```

Retorna as partições da rede

```
bipartite.sets(B)
```

Projetando uma rede bipartite

Podemos projetar uma rede bipartite para mostrar como um dos conjuntos de dados estão conectados. Na rede de exemplo, podemos mostrar como os fãs estão conectados de acordo com os seus times de preferência.

```
X = set(['A','B','C','D','E'])
P = bipartite.projected_graph(B, X)
```

Também podemos inserir pesos nas edges da rede projetada:

```
X = set(['A','B','C','D','E'])
P = bipartite.weighted_projected_graph(B, X)
```

hacking analytics

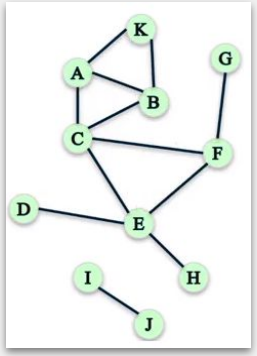
https://hackinganalytics.com/

Fonte: Applied Social Network Analysis in Python - University of Michigan

Clustering Coefficient

Mede o grau (degree) em que o nó de uma rede tende a formar clusters ou triângulos.

Rede de exemplo:



import networkx as nx
G = nx.Graph()
G.add_edges_from([(‘A’,’K’),
(‘A’,’B’), (‘A’,C’), (‘B’,’C’), (‘B’,’K’),
(‘C’,’E’), (‘C’,’F’), (‘D’,’E’), (‘E’,’F’),
(‘E’,’H’), (‘F’,’G’), (‘I’,’J’)])

Local Clustering Coefficient

Percentual de pares de conexões formados pelas conexões de um determinado node.

Exemplo: Calculando Local Clustering Coefficient do node C:

- Denominador
  Número de **possíveis** pares que as conexões de C conseguem formar entre si
  - Formula = (Dc(Dc - 1)) / 2
  - Onde Dc = Degree C = número de conexões = 4
  - Colocando na fórmula
    (4(4 - 1)) / 2 = 12 / 2 = 6
  - Que são os seguintes pares
    {[A,B], [E,F], [B,F], [A,F], [A,E], [B, E]}
- Numerador
  Número de pares **formados** entre as conexões de C
  - 2 = {[A,B], [E,F]}
- Resultado
  - 2/6 = 0,33
  - 33% de todos os possíveis pares das conexões de C foram efetivamente formados.
- Exceções
  - Quando nodes possuem menos do que duas conexões podemos ter uma divisão por zero durante o cálculo. Para estes casos Local Clustering Coefficient = 0

nx.clustering (G, ‘C’)
Resultado: 0,33

Global Clustering Coefficient

Abordagem 1: Average clustering
Calcular a média dos Local Clustering Coefficient de cada node.

nx.average_clustering (G)
Resultado: 0,28

Abordagem 2: Transitivity
Taxa de triângulos fechados e triângulos abertos dentro de uma rede.
A grande diferença entre average clustering e transitivity e é que esta última atribui mais peso aos nodes com maior degree.

Transitivity = (3*Número de triângulos fechados) / Número de triângulos abertos

nx.transitisity (G)
Resultado: 0,4090

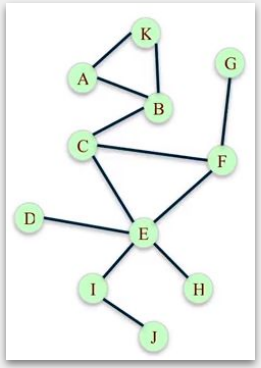
hacking analytics

https://hackinganalytics.com/
Fonte: Applied Social Network Analysis in Python - University of Michigan

Distance Measure

A **distância entre dois nodes** é dada pelo caminho (path) de **menor** tamanho entre eles.

Rede de exemplo:



A distância entre A e H é 4 (A-B-C-E-H)
nx.shortest_path_length (G)
Resultado: {'A': 0, 'B': 1, 'C': 2, 'D': 4, 'E': 3, 'F': 3, 'G': 4, 'H': 4, 'I': 4, 'J': 5, 'K': 1}

Eccentricity do node "n" é dada pela **maior** distância entre o node "n" e todos os demais nodes da rede

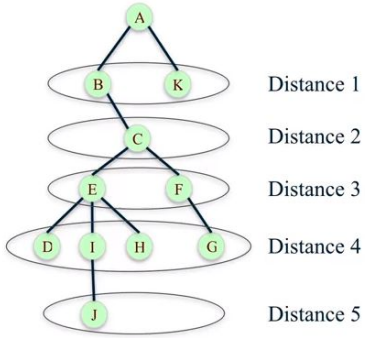
nx.eccentricity (G)
Resultado: {'A': 5, 'B': 4, 'C': 3, 'D': 4, 'E': 3, 'F': 3, 'G': 4, 'H': 4, 'I': 4, 'J': 5, 'K': 5}

Calculando a distância entre dois nodes de forma sistemática

Breadth-first search
Separa os nodes em camadas para encontrar as distâncias de um determinado node para os demais nodes da rede.

Exemplo: calcular a distância entre o node A e os demais nodes da rede.

1. A primeira camada possui todas as conexões do node A.
2. A segunda camada possui todas as conexões dos nodes da camada anterior, excluindo aqueles que já foram selecionadas anteriormente
3. Seguir passo 2 até até acabar os nodes



Montar a árvore acima:
T = nx.bfs_tree (G, 'A')
Extrait os pares da árvore:
T.edges()

Informações gerais de distância da rede

Average distance
Distância média entre todos os pares de nodes.
nx.average_shortest_path_length (G)
Resultado: 2,52

Diameter
Maior distância entre qualquer par de nodes
nx.diameter (G)
Resultado: 5

Radius
O menor valor de eccentricity identificado na rede.
nx.radius (G)
Resultado: 3

Identificando nodes centrais (center) e periféricos (peripheral)

Periphery é o conjunto de nodes que cujo eccentricity = diameter.
nx.periphery (G)
Resultado: ['A', 'K', 'J']

Center é o conjunto de nodes que cujo eccentricity = radius
nx.center (G)
Resultado: ['C', 'E', 'F']

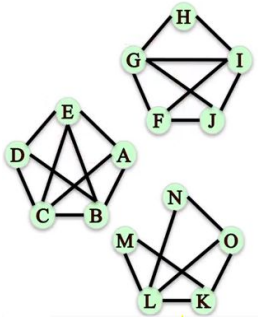
hacking analytics

https://hackinganalytics.com/
Fonte: Applied Social Network Analysis in Python - University of Michigan

Connected Components

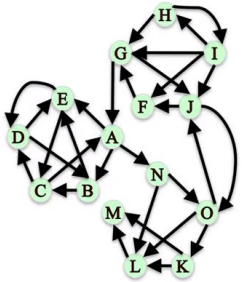
Dizemos que a rede está conectada quando existe um caminho para cada par de nodes.

Rede de exemplo:



nx.is_connected (G)
Resultado: False

Exemplo de directed graph:



Identificando connected components dentro da rede

Dizemos que um subconjunto de nodes está conectado se:

- Existe um caminho para cada par de nodes.
- Nenhum outro node possui um caminho válido para os nodes do subconjunto em questão

nx.number_connected_components (G)
Resultado: 3

nx.connected_components (G)
Resultado: [{'A','B','C','D','E','F'}, {'G','H','I','J'}, {'K','L','M','N','O'}]

nx.connected_component (G, 'M')
Resultado: {'K','L','M','N','O'}

Connected components dentro de directed graphs

Dizemos que a rede é **strongly connected** quando existe um caminho cada cada par de nodes em ambos os sentidos (exemplo: de A para B e também de B para A).

nx.is_strongly_connected (G)
nx.strongly_connected_components (G)

Também podemos transformar uma rede "directed" em "undirected". Neste caso passam a valer as regras de connected components para redes undirected.

Se a sua rede versão "undirected" estiver conectada, então a classificamos como **weakly connected**.

nx.is_weakly_connected (G)
nx.weakly_connected_components (G)

Network Robustness

A idéia é que redes com maior node connectivity e edge connectivity são mais robustas, já que a rede será menos sensível (tende a não mudar) a remoção de nodes e edges.

Node connectivity
Número mínimo de nodes necessário para desconectar a rede.

nx.node_connectivity (G)
nx.minimum_node_cut (G)
Retorna o nodes que, se removidos, desconectariam a rede.

Edge connectivity
Número mínimo de edges necessário para desconectar a rede.

nx.edge_connectivity (G)
nx.minimum_edge_cut (G)

Todas as funções acima também aceitam os parâmetros (G, 'A', 'B'). Neste caso, estamos analisando a desconexão entre os nodes 'A' e 'B'.

Simple path
nx.all_simple_paths(G, 'G', 'L')
Retorna uma lista de possíveis paths que conectam 'G' e 'L'.

hacking analytics

https://hackinganalytics.com/
Fonte: Applied Social Network Analysis in Python - University of Michigan

Degree Centrality

O degree de um node é dado pelo seu número de conexões.
Em redes direcionadas, também podemos encontrar as variações “in degree” (número de conexões que chegam ao node) e “out degree” (número de conexões de saem do node).

A métrica **degree centrality** é dada pela equação dc = dv / (n-1), onde “dv” é o degree do node “v” e “n” é o número total de nodes na rede. **Ou seja, esta métrica atribui mais importância aos nodes com maior número de conexões.**

```
G = nx.karate_club_graph()
G = nx.convert_node_labels_to_integers(G,
first_lable = 1)
dc = nx.degree_centrality(G)
```

Para calcular “in degree centrality” e “out degree centrality”:

```
dc = nx.in_degree_centrality(G)
dc = nx.out_degree_centrality(G)
```


Fonte: Applied Social Network Analysis in Python - University of Michigan

Closeness Centrality

Esta métrica assume que nodes importantes estão perto de outros nodes.Seu cálculo é dado pela equação *cc_node_v = (n-1) / soma do shortest path length do node v contra os demais nodes da rede*. “n” é o número total de nodes na rede. Perceba que o maior valor para esta métrica é 1.

cc = nx.closeness_centrality(G)

Você pode extrair os valores do numerador e denominador usados para calcular a closeness centrality. Tomando como exemplo o node 32:

numerador = sum(nx.shortest_path_length(G, 32).values())
denominador = len(G.nodes())-1

Closeness Centrality em redes desconectadas

Temos algumas opções para medir closeness centrality em redes desconectadas.

Uma das opções é considerar apenas os nodes que podem ser alcançados pelo no que desejamos calcular closeness centrality.

Neste caso, podemos alterar a equação que define esta métrica para equação *cc_node_v = (n-1) / soma do shortest path length do node v contra os demais nodes que v consegue alcançar.* “n” é o número total de nodes na rede que o node v consegue alcançar.

cc = nx.closeness_centrality(G, normalized=False)

Algumas vezes esta abordagem pode ser problemática, pois em um cenário em que um node “v” consegue se conectar apenas com um node em uma rede direcionada (e esta conexão de fato existe), teríamos uma closeness centrality igual a 1 (maior valor possível) para um node que não parece tão importante na rede.

Para resolver este problema podemos utilizar uma alternativa a primeira opção descrita, na qual temos que normalizar o resultado pelo percentual de nodes que o node “v” pode alcançar em toda a rede. Na prática, utilizamos a mesma fórmula descrita na opção 1, porém multiplicamos o resultado por:
Número de nodes que ”v” consegue alcançar / número total de nodes - 1.

cc = nx.closeness_centrality(G, normalized=True)

Fonte: Applied Social Network Analysis in Python - University of Michigan

Betweenness Centrality

Esta métrica assume que nodes importantes são aqueles que conectam outros nodes. Suponha que queremos calcular a betweenness centrality do node v. Para isso, precisamos calcular a taxa a seguir para cada par de nodes s, t:

Numerador: Número de **shortest paths** entre os nodes s, t
Denominador: Número de shortest paths entre os nodes s, t que passam pelo node v

O resultado final será dado pela soma destas taxas para todas as possibilidades de combinações entre nodes s, t.
Adicionalmente, temos a opção de considerar ou não o próprio node v entre as combinações de s, t. Quando incluímos o node v nas possíveis combinações de s,t teremos uma betweenness centrality maior (parâmetro endpoints no exemplo mais abaixo).

Normalização
Redes maiores tendem a ter um valor de betweenness centrality maior já que possuem mais nodes. Portanto, se queremos comparar esta métrica em diferentes redes precisamos normalizar o resultado. Para isso devemos dividir o resultado pelo número de pares na rede (excluindo node v). Esse valor é dado por:
Redes não direcionadas = 0.5(N-1)(N-2)
Redes direcionadas = (N-1)(N-2)
Onde N é o número total de nodes da rede.

Aproximação
Calcular betweenness centrality em redes muito grandes pode exigir muito poder computacional. Ao invés de usar todos os nodes da rede para calcular a métrica, podemos trabalhar com apenas uma amostra de nodes (parâmetro k no exemplo abaixo).

bc = nx.betweenness_centrality(G, normalized=True, endpoints=False, k=10)

Subsets
Podemos calcular esta métrica em conjuntos específicos de nodes.Devemos informar duas listas de nodes (source e target), sendo que os nodes s,t serão selecionados sempre um de cada lista.

bc = nx.betweenness_centrality_subset(G, [lista de source nodes], [lista de target nodes], normalized=True)

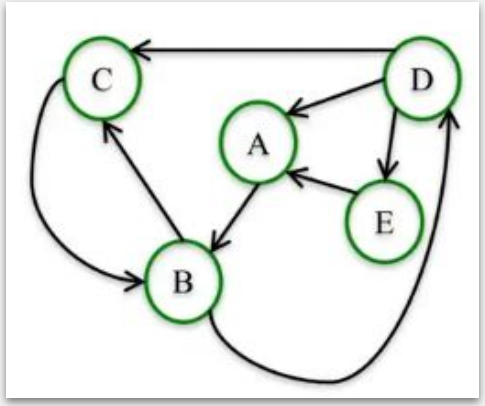
Edges
Finalmente podemos calcular betweenness centrality de edges.

bc = nx.edge_betweenness_centrality(G, normalized=True)

Page Rank

Esta métrica foi criada pelo Google para medir a importância de páginas web. Page Rank assume que nodes importantes são aqueles que possuem muitas conexões de entrada (in-links) de páginas importantes.
Ou seja, esta métrica é mais útil em redes direcionadas.

Rede de exemplo:



Etapas para calcular Page Ran:

1. Todos os nodes começam com o valor 1/n, onde n é o número de nodes da rede
2. Calcular Page Rank k vezes

Passo a passo

Vamos realizar o passo a passo para calcular Page Rank para a rede de exemplo ao lado, usando k = 1.

Para k = 0 temos:

A	B	C	D	E
1/5	1/5	1/5	1/5	1/5

Para k = 1, vamos atualizar apenas o valor de A como exemplo:

1. Selecionar todos os nodes que A recebe conexão (D, E)
2. Calcular Page Rank que A vai receber de cada node
   a. Page Rank node D
      D aponta para três nodes (A, C, D), ou seja, A recebe ⅓ das conexões de D
      Portanto temos (1/5) * (1/3), onde (1/5 é o valor antigo do Page Rank de D)
   b. Page Rank node E
      E aponta apenas para o node A, ou seja, A recebe 1/1 das conexões de E
      Portanto temos (1/5) * (1/1), onde (1/5 é o valor antigo do Page Rank de E)
3. Somar sub totais do passo 2
   4/15

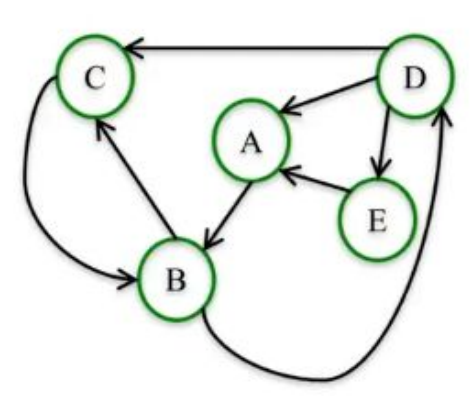
Depois de muitas iterações os valores de Page Rank de cada node tende a estabilizar, assim temos os valores finais da métrica.

hacking analytics

https://hackinganalytics.com/
Fonte: Applied Social Network Analysis in Python - University of Michigan

Interpretação de Page Rank

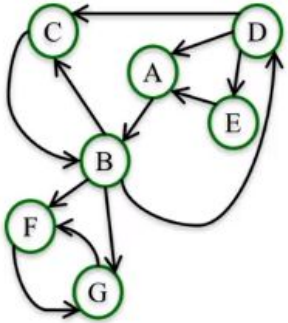
Dada a rede de exemplo abaixo, veja que podemos caminhar de node em node seguindo a direção das edges. A partir do node D posso acessar o node E (passo 1), que pode acessar o node A (passo 2), que pode acessar o node B (passo 3), e assim por diante (passo n).
Note também que um determinado node eventualmente poderá acessar mais de um node, como no caso do node D que pode acessar C, A, E. **Random Walk** é um termo utilizado quando dizemos que os nodes serão acessados de forma aleatória na rede.



Explicado isso, podemos interpretar o valor de Page Rank de cada node como a probabilidade de, após uma caminhada de k passos aleatórios, terminarmos a caminhada naquele node

Scaled Page Rank

Levando em consideração a abordagem de random walks explicada anteriormente, em alguns tipos de redes podemos ter problemas com Page Rank, especificamente em casos em que os nodes se conectam de certa forma que formam um loop infinito entre eles.



Nesta rede ao lado, se considerarmos um número grande de k passos para caminhar na rede, claramente podemos ver que a probabilidade de pararmos nos nodes F ou G será de 50% para cada um, e os demais nodes terão 0% de probabilidade (estes serão o valor de Page Rank).

Podemos resolver este problema adicionando um “damping parameter” alpha. Este parâmetro faz com que, durante a caminhada entre os nodes, um node seja escolhido aleatoriamente x% das vezes (onde x será dado por 1-alpha)

Portanto, interpretamos Scaled Page Rank como a probabilidade de, após uma caminhada de k passos aleatórios e com a introdução de um damping parameter, terminarmos a caminhada naquele node.

Em geral, configuramos o valor do damping parameter entre 0,8 e 0,9, ou seja, na maioria das vezes iremos seguir a orientação da rede durante a caminhada na rede; porém, de 10 a 20% das vezes escolheremos um node aleatório sem seguir a direção das edges.

Scaled Page Rank costuma funcionar melhor em redes muito grandes.

pagerank(G, alpha=0.8)

Fonte: Applied Social Network Analysis in Python - University of Michigan

Hubs e Authorities

Assim como Page Rank, esta métrica também é mais utilizada e para encontrar páginas web importantes. Tomando como exemplo um portal da internet, **authorities** serão aquelas páginas principais e **hubs** serão as páginas que conectam estas páginas importantes.
Neste abordagem, chamamos de **base set** o conjunto de nodes considerados authorities ou hubs**.**

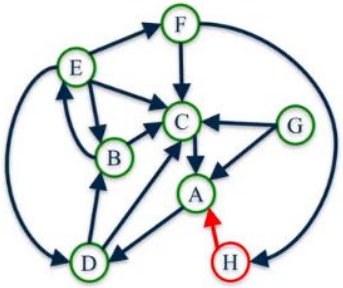
HITS Algorithm
Este é um algoritmo utilizado para calcular hubs e authorities em uma rede. Passo a passo para calcular o valor de hub e authority do node v:

1) Iniciar cada node com ambos de de authority e hub igual a 1
2) Aplicar a regra para atualizar o valor de authority (somar valor de hub de cada node que aponta para v - in degree)
3) Aplicar a regra para atualizar o valor de hub (somar valor de authority de cada node que v está apontando - out degree)
4) Normalização
Para normalizar o valor de authority de cada node devemos a) somar o valor de authority de todos os nodes e b) dividir o valor de authority de cada node por (a). Usamos a mesma regra para normalizar o valor de hub, porém usando os valores de hub.
5) Repetir o processo k vezes

hits(G)

Exemplo

Levando em consideração a rede abaixo, iremos calcular o valor de authority e hub de cada node (em uma única iteração, k=1).



- Node A possui authority igual a 3/15 na primeira iteração porque A recebe conexão de C, G, H. Cada um desses nodes possui hub igual a 1. Portanto, authority score de A será a soma dos hub scores de C, G, H.
- Se somarmos os valores da coluna “New Auth” teremos 15. Este foi o valor usado para normalizar o resultado final.

Antes da normalização

	Old Auth	Old Hub	New Auth	New Hub
A	1	1	3	1
B	1	1	2	2
C	1	1	5	1
D	1	1	2	2
E	1	1	1	4
F	1	1	1	2
G	1	1	0	2
H	1	1	1	1

Depois da normalização

	Old Auth	Old Hub	New Auth	New Hub
A	1	1	3/15	1/15
B	1	1	2/15	2/15
C	1	1	5/15	1/15
D	1	1	2/15	2/15
E	1	1	1/15	4/15
F	1	1	1/15	2/15
G	1	1	0/15	2/15
H	1	1	1/15	1/15

Fonte: Applied Social Network Analysis in Python - University of Michigan

Preferential Attachment Model

É um modelo que simula a criação de uma rede que segue a distribuição **power law**. Nesta distribuição existem poucos nodes com muitas conexões e vários nodes com poucas conexões.

Neste modelo o novo node é anexado à rede de forma aleatória, porém respeitando uma probabilidade de "attachment" associada a cada node.

Esta probabilidade é dada pelo número de conexões do node (degree) dividido pelo número total de nodes na rede.

```
G = nx.barabasi_albert_graph(1000000, 1)
degrees = G.degree()
degree_values = sorted (set (degrees.values()))
histogram =
[list(degrees.values().count(i))/float(nx.number_of
_nodes(G)) for i in degree_values]

plt.plot(degree_values, histogram, 'o')
plt.xlabel('Degree')
plt.ylabel('Fraction of Nodes')
plt.xscale('log')
plt.yscale('log')
plt.show()
```

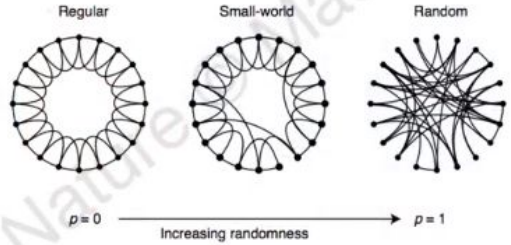
Small World Network

Este modelo simula a criação de uma rede com as seguintes premissas: Redes têm tendência a possuir **alto clustering coefficient** e **baixo average path length**.

Preferential Attachment Model possui baixo average path length, mas **não possui** alto valor de clustering coefficient. Isso porque não existe nenhum mecanismo especial que força a criação de triângulos na rede.

Small World Network funciona da seguinte forma:
1) Começamos com uma rede no formato de um anel, na qual cada node está conectado com os seus vizinhos mais próximos
2) Configurar o parâmetro "p" entre 0 e 1. Este parâmetro define a probabilidade de realizarmos ajustes em cada edge da rede.
3) Redefinir as edges existentes na rede de acordo com a probabilidade de "p".

Exemplo: p = 0.4, ou seja, 40% das edges da rede serão modificadas aleatoriamente. Se considerarmos que uma edge conecta os nodes A e B, e decidirmos que esta edge será ajustada, significa que o node A será conectado a outro node da rede de forma aleatória.



Watts and Strogatz, 1999

Veja o que acontece para os diferentes valores de p.
- Quando p = 0, significa que teremos uma rede sem nenhum ajuste de edge, resultando em alto clustering coefficient (vários triângulos locais), porém alto average path.
- Quando p = 1, significa que ajustaremos todas as edges da rede, resultando em baixo clustering coefficient (poucos triângulos locais) e baixo average path.
- Alcançaremos o modelo que queremos com valor p pouco maior do que 0. Neste caso ajustaremos algumas edges e teremos as duas propriedades desejadas (alto clustering coefficient e baixo average path length).

watts_strogatz_graph (n, k, p)

Onde n é o número de nodes que serão criados, k é a quantidade de vizinhos mais próximos de cada node serpa conectado, e p é o parâmetro de probabilidade de ajuste de cada edge.

connected_watts_strogatz_graph (n, k, p, t)

Para garantir que no final ainda teremos uma rede conectada. Executa *watts_strogatz_graph (n, k, p) t vezes até conseguir uma rede conectada.*

hacking analytics

https://hackinganalytics.com/
Fonte: Applied Social Network Analysis in Python - University of Michigan

Link Prediction

Prevê quais são as possíveis conexões que poderiam ocorrer na rede. Nos exemplos a seguir vamos analisar diferentes métricas para quantificar a chance de X se conectar com Y.

Common Neighbors

Número de nodes em comum entre os nodes X e Y.

cn = [(e[0], e[1], len(list(nx.common_neighbors(G, e[0], e[1])))) for e in nx.non_edges(G)]

Então poderíamos dizer quem são os nodes com maior chances de se conectar, de acordo com o número de conexões em comum que os mesmos compartilham.

Jaccard Coefficient

Número de nodes em comum entre os nodes X e Y normalizado pelo número (distinto) de nodes conectados com X *ou* Y. Ou seja, é o percentual das conexões de X e Y que são compartilhadas entre os dois nodes.

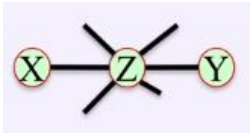
j = list(nx.jaccard_coefficient(G))

Resource Allocation Index

Taxa de "resource" que um node pode enviar para outro node por meio de suas conexões em comum. Calculamos esta métrica da seguinte forma:

Cada cada node Z em comum entre X e Y, fazemos: 1 / degree Z. Então somamos todos os valores parciais.

ra = list(nx.resorce_allocation_index(G))



No exemplo ao lado, X e Y se conectam por meio de Z. Porém, Z se conecta com outros 4 nodes. Portanto, quando X envia uma mensagem para Y por meio de Z, 1/5 é a taxa de resource que Z enviará para Y. Em outras palavras, esta métrica penaliza nodes em comum que, por sua vez, possuem muitas conexões

Adamic-Adar Index

Muito similar ao Resource Allocation Index, porém ao invés de dividir pelo degree dp node em comum Z, a divisão é feita pelo log (Z).

ra = list(nx.adamic_adar_index(G))

Preferential Attachment Score

Nodes com mais conexões (alto degree) possuem maior score. A métrica é calculada pela multiplicação dos valores de degree dos nodes X e Y.

pa = list(nx.preferencial_attachment(G))

Community Common Neighbors

Usada para casos em que a rede possui o conceito de comunidades (por exemplo, pessoas que trabalham no mesmo departamento).

Esta métrica assume que nodes que pertencem à mesma comunidade têm mais chance de se conectarem.

Calculamos a métrica entre os nodes X e Y da seguinte forma:

Número de nodes em comum entre X e Y + Número de nodes em comum entre X e Y que pertencem à mesma comunidade.

```
# Define as comunidades
G.node['X']['community'] = 0
G.node['Y']['community'] = 1
# Calcula a métrica
ccn= list(nx.cn_soundarajan_hopcroft(G))
```

hacking analytics

https://hackinganalytics.com/

Fonte: Applied Social Network Analysis in Python - University of Michigan

Link Prediction

Community Resource Allocation
Similar à métrica Resource Alcoation Index, mas leva em consideração **apenas os nodes da mesma comunidade**.
cra = list(nx.ra_index_soundarajan_hopcroft(G))

Em geral, estas métricas nos darão diferentes visões de como poderíamos prever a conexão entre os nodes. Você pode usar todas estas métricas como features de um modelo de classificação supervisionado para resolver o problema de **link prediction**.

Fonte: Applied Social Network Analysis in Python - University of Michigan