Catalogação de funcionalidades

Jefter Santiago

15 de março de 2023

Conteúdo

Em geral vou me referir quando me referir a grafos estou falando sobre o modelo matemático e redes esse modelo aplicado a algum dado. Quando me cito **funcionalidades** de algum pacote me refiro a algoritmos de grafos.

Nessa etapa do projeto foram analisadas as funcionalidades dos pacotes e suas respectivas abrangências. Estas são relacionadas a `geração` de grafos, medidas de `transitividade, centralidade, conectividade` etc.

As bibliotecas possuem algoritmos mais básicos de grafos como os de busca e para percorrer grafos, mas nessa catalogação só listei os que considero importantes para trabalhar com redes complexas. Como os algoritmos de medidas de centralidade, conectividade etc.

Alguns algoritmos podem ter ficado de fora desse catalogo, isso decorre do motivo citado antes. Nem todos algoritmos presentes nos pacotes são relevantes para o uso mais básico deles. Essa lista é pensada para pessoas iniciantes no estudo de redes complexas e que portanto devem utilizar essas ferramentas mais "relevantes"no inicio da jornada. A partir dessas funcionalidades é possível aprender a utilizar as muitas outras presentes em cada um dos pacotes de acordo com a necessidade do que está tentando fazer. Porém, com base na minha experiência acredito que essa lista contém mais que o necessário para usuários iniciantes.

1 NetworkX

1 Texto inicial sobre as funcionalidades

Esse pacote me parece ser o mais didático, sobretudo para pessoas que estão iniciando seus estudos em grafos ou redes complexas. Ser escrito em Python contribui para a maior facilidade de se aprender a utiliza-lo. A documentação é no geral mais do que suficiente para se aprender, é bem divida e fácil de encontrar as informações. Além disso o projeto do NetworkX de incentivar contribuidores a escrever `notebooks` com explicações de algoritmos parece ser uma ideia interessante, ainda que, no momento, possuam poucos notebooks com explicações.

Para tarefas mais "avançadas"o pacote pode não apresentar eficiência tão boa quanto os outros, e isso talvez se deve ao fato de ser escrito puramente em Python - isso o deixa mais leve (ocupação do disco), praticidade e simplicidade da linguagem - o faz ter alguns problemas que a própria linguagem tem. Em relação aos outros pacotes (escritos em C ou C++) o NetworkX apresenta a menor eficiência na execução de algoritmos. **Isso foi constatado quando fiz os testes com algumas funcionalidades do catalógo. E note que elas foram testadas para todo tipo e tamanho de rede e várias vezes.**

Com isso, entendo que o pacote pode não ser o melhor para trabalhos que envolvam redes muito grandes (com muitos vertices e arestas) pois na execução de alguns algoritmos em redes dessa proporção o NetworkX pode exigir muito uso de memória e/ou espaço, **como é o caso do algoritmo de Dijkstra testado para várias redes de vários tamanhos**.

Pode ser um bom pacote para se ter um primeiro contato no estudo de redes complexas, já que é mais leve, de simples instalação e simples de usar - principalmente para pessoas já experientes com Python. Porém, para trabalhos de cunho mais aprofundado deixa a desejar.

1 Notas adicionais

- Falar sobre a visualização dos grafos usando esse pacote depender do matplotlib, se é bom ou ruim e quais são as limitações do matplotlib e como isso impacta o NetworkX.
-

1 Catalogo

1.3.1 Criação de grafos

`fazer tabela` dirigidos e não dirigidos, ponderados e não ponderados, além de grafos bipartidos.

Os grafos podem ser criados também a partir de matrizes de adjacências ou listas de adjacências e arquivos `gml`.

1.3.2 Algoritmos de caminho mínimo

Implementação do pacote	Descrição da função
`dijkstra_path()`	Retorna caminho mais curto do entre dois vertices em um grafo.
`astar_path()`	Retorna lista com vertices do caminho mais curto de um grafo.
`bellman_ford_path()`	Retorna caminho mínimo de um vertice a outro em um grafo ponderado.
`johnson()`	
`all_pairs_dijkstra_path()`	
`floyd_warshall()`	
`bidirectional_dijkstra()`	

1.3.3 Algoritmos de busca de caminhos

Implementação do pacote	Descrição da função
`all_simple_paths()`	
`all_shortest_paths()`	
`shortest_simple_paths()`	
`single_source_shortest_path()`	

1.3.4 Algoritmos de conectividade

Implementação do pacote	Descrição da função
`connected_components()`	
`node_connected_components()`	
`strongly_connected_components()`	
`weakly_connected_components()`	

1.3.5 Algoritmos de árvore geradora

Implementação do pacote	Descrição da função
`minimum_spanning_tree()`	
`maximum_spanning_tree()`	
`all_spanning_trees()`	

1.3.6 Medidas de centralidade

Implementação do pacote	Descrição da função
`degree_centrality()`	
`closeness_centrality()`	
`betweenness_centrality()`	
`eigenvector_centrality()`	
`age_rank()`	
`katz_centrality()`	

1.3.7 Algoritmos de detecção de comunidade

Implementaçõ do pacote	Descrição da função
`girvan_newman()`	
`louvain_communities()`	
`label_propagation_communities()`	
`asyn_fluidc()`	

1.3.8 Geradores de grafos

- Grafos padrão

Implementação do pacote	Descrição da função
`complete_graph()`	
`complete_bipartite_graph()`	
`cycle_graph()`	
`path_graph()`	
`star_graph()`	
`wheel_graph()`	

- Grafos aleatórios

Implementação do pacote	Descrição da função
`erdos_renyi_graph()`	
`gnm_random_graph()`	
`dense_gnm_random_graph()`	
`barabasi_albert_graph()`	
`watts_strogatz_graph()`	
`connected_watts_strogatz_graph()`	
`random_regular_graph()`	
`random_lobster()`	
`powerlaw_cluster_graph()`	

- Grafos de árvores

Implementação do pacote	Descrição da função
`random_tree()`	
`uniform_random_tree()`	
`mst_random_tree()`	
`minimum_spanning_tree()`	

- Grafos de treliças

Implementação do pacote	Descrição da função
`grid_graph()`	
`hypercube_graph()`	
`torus_graph()`	

- Redes sociais

Implementação do pacote	Descrição da função
`karate_club_graph()`	
`florentine_families_graph()`	
`davis_southern_women_graph()`	

2 igraph

3 graph-tools