Implementatie van de Louvain Method voor het deteceteren van netwerken in een gewogen graaf

Taak 1 Algoritmen en datastructuren 2

Abdullah Sabaa Allil - 0575278

abdullah.sabaa.allil@vub.be

2021-2022

Motivatie van keuzes

Bij het berekenen van de modulariteit in de procedure *modularity* wordt er nagegaan of de waarde van de delta functie niet nul is. Als de waarde nul is, dan hoeven we die berekeningen niet uit te voeren voor deze bepaalde term en kunnen we dus efficienter de modulariteit vinden.

Voor het dictionary die de intra-degrees bevat en die teruggegeven moet worden na de eerste fase wordt er voor de avl implementatie gekozen. Deze implemenatie is performant, alle werkpaarden (waaronder het zoeken) van het dictionary ADT zijn in log(n). De avl implementatie zal ook niet meer geheugen gebruiken dan nodig, aangezien het toelaat om een variabel aantal elementen bij te houden en geen maximum waarde heeft van het aantal elementen dat kunnen bijgehouden worden.

In de tweede fase van het algoritme wordt het aantal communities berekend door de avl boom die de dictionary voorstelt te traversen met een iteratief tree traversal algoritme. Hiervoor wordt iterative-pre-order gebruikt. De boom traversen is efficienter dan de vector van de community id's te traversen aangezien we minder communities hebben dan nodes in de graaf. Om een voorbeeld te geven: In de eerste pass van het algoritme uitgevoerd op de Blondel graaf hebben we 16 nodes en 4 communities. We kunnen de node-indexed vector van community id's gaan aflopen en aan de hand daarvan het aantal communities berekenen, maar het is efficienter om de boom te gaan traversen, en maar 4 elementen bezoeken.

In de tweede fase van het algoritme wordt er een vector en een nieuwe dictionary met opnieuw een avl-implementatie aangemaakt: de vector is namelijk een node-indexed vector, waarbij de indices de nodes voorstellen van de nieuwe geaggregeerde graaf. De inhoud van de vector is dan de communities. De dictionary heeft de communities als keys, en de nodes van de nieuwe graaf als values. Beide worden gebruikt voor het efficient zoeken in twee richtingen: wanneer het nodig is om te weten welke community zit in een bepaalde node in de nieuwe graaf en wat het gewicht is van deze community, dan wordt de vector gebruikt, en is het zoeken O(1). Wanneer het nodig is om te weten in welke node een bepaalde community zit, wordt de avl dictionary gebruike, dan is het zoeken O(log(n)).

Voor het maken van die vector en van de avl boom wordt er opnieuw gebruikgemaakt van iterative-pre-order tree traversal algoritme om de dictionary met de communities en hun gewichten te traversen.

De vector wordt gebruikt om de zelfbogen van de nodes van de geaggregeerde graaf te genereren. Er wordt dan telkens gezocht naar het gewicht van de community en dat gewicht wordt meegegeven als gewicht van de zelfboog. De dictionary wordt dan gebruikt om teweten of er bogen zijn tussen de nodes die de communities voorstellen in de nieuwe geaggregeerde graaf.

Aanpassingen aan de a-d folder

Er wordt gebruikgemaakt van de volgende bestanden in de a-d folder:

- ***a-d/dictionary/ordered/avl.rkt*** Dit is de avl implementatie van het dictionary ADT.
- ***a-d/tree/binary-tree-algorithms*** Dit bestand bevat de algoritmen om een boom te traversen. In de geïmporteerde libraries wordt (a-d tree avl-tree) geïmporteerd in plaats van (a-d tree binary-tree). Dit is nodig aangezien we een avl boom willen traversen in onze implementatie van de Louvain method.
- ***a-d/tree/avl-tree*** Dit bestand is nodig om de root van de avl boom te kunnen nemen en meegeven aan het algoritme dat de avl boom gaat traversen.

Onderstaande library (bestand) wordt geimplementeerd en toegevoegd:

- ***a-d/displaying.rkt*** Dit is een library die de resultaten van het algoritme(voor zowel de eerste als de tweede fase) displayt. De belangrijkste procedures hieruit zijn:
    - *display-first-phase (number vector<community-id> dictionary -> void)* om resultaten van de eerste fase (van *detect-communities*) te displayen;
    - *display-louvain (list string -> void)* om de resultaten van het louvain algoritme te displayen. Deze procedure verwacht een lijst, namelijk het resultaat van de procedure *louvain* uit te voeren, en een string, namelijk de naam van de graaf;
    - *display-graph (weighted-graph -> void)* om een graaf te displayen op het scherm (de bogen tussen de nodes en de corresponderende gewichten).