**Feladatkiírás**

Szociális hálózatok (gráfok) gyakran közösségi szerkezetűek. Ezek olyan részgráfok, amelyekben a csúcsok egymás között sűrűbben összekötöttek, mint kifelé, másik közösségek felé. Ez tehát a klaszterezés egy gyengített definíciója. Rengeteg megoldási módszer született már. Ezek közül néhány lineáris programozási modellre épülő eljárás. A hallgató feladata legalább két ilyen modell tanulmányozása, implementációja és összehasonlító tesztelése.

**Tartalmi összefoglaló**

**A téma megnevezése:**

Algoritmusok vizsgálata hálózati közösségek meghatározására

**A megadott feladat megfogalmazása:**

**A megoldási mód:**

**Alkalmazott eszközök, módszerek:**

**Elért eredmények:**

**Kulcsszavak:**

hálózat, közösségkeresés, modularitás, lineáris programozás, utazó ügynök probléma

**Tartalomjegyzék**

[Feladatkiírás 2](#_Toc80443487)

[Tartalmi összefoglaló 3](#_Toc80443488)

[Tartalomjegyzék 4](#_Toc80443489)

[1. Bevezetés 5](#_Toc80443490)

[2. Alapfogalmak 5](#_Toc80443491)

[2.1. Gráfelméleti alapfogalmak 5](#_Toc80443492)

[2.2. Gráfreprezentációk 5](#_Toc80443493)

[2.3. Sztochasztikus mátrixok 5](#_Toc80443494)

[2.4. Közösségek gráfokban 5](#_Toc80443494)

[2.5. Modularitás 5](#_Toc80443494)

[3. eljárások 5](#_Toc80443495)

[3.1. IP alapú 5](#_Toc80443496)

[3.1.1. Modularitás maximalizálási probléma 5](#_Toc80443498)

[3.1.2. Egészértékű lineáris programozási modell leírása 5](#_Toc80443499)

[3.2. TSP alapú 5](#_Toc80443497)

[3.2.1. PRF mátrix: PageRank Feature 5](#_Toc80443498)

[3.2.2. PRD mátrix: PageRank Distance 5](#_Toc80443499)

[3.2.3. TSP solver 5](#_Toc80443500)

[3.2.4. Automatikus küszöbérték alapú vágás 5](#_Toc80443500)

[4. tesztelés 5](#_Toc80443501)

[4.1. NMI pontosság 5](#_Toc80443502)

[4.2. Alapigazság gráfok 5](#_Toc80443503)

[4.3. Gráfgyár 5](#_Toc80443504)

[4.1. IP alapú eredmények 6](#_Toc80443506)

[4.3. TSP alapú eredmények 6](#_Toc80443508)

[4.5. Eredmények összehasonlítása 6](#_Toc80443507)

[5. összefoglalás 6](#_Toc80443505)

[Irodalomjegyzék 7](#_Toc80443509)

[Nyilatkozat 8](#_Toc80443510)

[Köszönetnyilvánítás 9](#_Toc80443511)

**1. Bevezetés**

**Ezt írd meg a legvégén**

**Indítsd messziről (gráfok mindenütt, stb). Itt nem írj le konkrét eredményeket, csak vázold, hogy mi fog történni**

**2. Alapfogalmak**

**2.1. Gráfelméleti alapfogalmak**

Egy *G* = (*V*, *E*) *gráf* *V*={1, 2, …, *n*} *csúcsok* halmazából és a csúcsokat összekötő *E* ⊆ *V*×*V* *élek* halmazából áll. A gráf élét meghatározó két csúcsot *végpontoknak* nevezzük. A gráf *mérete* az éleinek számával egyezik meg, a gráf pontjainak száma a gráf *rendje*.

Az él *hurokél*, ha a két végpont megegyezik. Ha több él ugyanazt a csúcspárt köti össze, *többszörös élről* beszélünk. Az *egyszerű gráfokban* nincsenek többszörös élek és nincs hurokél. Ha a gráf minden éle rendezett csúcspár, akkor *irányított gráfnak* nevezzük. Az *irányítatlan gráfok* élei rendezetlen párok halmaza, (*i*,*j*) ∈ *V* esetében *i* pontból *j* pontba vezet él. Irányítatlan gráf ábrázolható irányítottként is úgy, hogy minden csúcspárt egy-egy él köt össze mindkét irányba. *Súlyozott gráf* esetén minden élhez egy valós számot rendelünk, ez a szám az él *súlya*.

A *G* = (*V*, *E*) egyszerű gráfot *teljes* *gráfnak* vagy *klikknek* nevezzük, ha bármely két különböző pontját pontosan egy él köti össze. Az éleinek száma: |E|=|V|(|V|-1)/2. A *G* = (*V*, *E*) gráf maximális mérete |V|(|V|-1)/2. *Sűrű gráfról* beszélünk, ha |E|~|V|2, a sűrű gráf éleinek száma közelíti a gráf maximális méretét. Egy gráfot *ritka* *gráfnak* nevezünk, ha az élek száma a gráf pontjai számának nagyságrendjében van, |E|~|V|.

Irányítatlan gráf *i* ∈ *V*csúcsának *fokszáma*a csúcsból induló élek száma, *di*-vel jelöljük. A gráf *fokszámsorozata* a gráf csúcsainak fokszámaiból álló lista, (*k1*, *k2*, …, *kn*). Az *i* és a *j* csúcs *szomszédok* vagy *szomszédosak*, ha összeköti őket él.

Egy *G’* = (*V’*, *E’*) gráf *G* = (*V*, *E*) gráf *részgráfja*, ha *V’* ⊆ *V* és *E’* ⊆ *E*. *Vágásnak* nevezzük *V* csúcshalmaz felosztását két (*S* és *V*-*S*) halmazra.

*G* gráfban v0, e1, v1, . . . , vk−1, ek, vk sorozatot *sétának* nevezünk, ha v0, v1, . . . , vk G csúcsainak sorozata, v1, . . . , vk G éleinek sorozata, és ei ∈ E két végpontja vi−1 és vi minden i ∈ {1, . . . , k} esetén, k a séta hossza. A sétát *útnak* nevezzük, a nincs benne élismétlés. Egy út *kör*, ha az első és az utolsó csúcs megegyezik. A *Hamilton-kör* olyan kör, amely a gráf minden pontját érinti.

A továbbiakban irányítatlan, súlyozatlan egyszerű gráfokat értünk gráf és hálózat kifejezések alatt.

**2.2. Gráfreprezentációk**

A gráfokat ábrázolhatjuk rajzolással, ekkor a pontok a gráf csúcsainak felelnek meg, a pontokat összekötő vonalak a gráf éleinek. A gráfalgoritmusokhoz azonban valamilyen adatszerkezetre lesz szükségünk, amiben a gráfokat tároljuk, és különféle eljárások különféle adatszerkezetet használnak. A következőkben összefoglaljuk ezeket előnyeikkel és hátrányaikkal.

Egy ilyen adatszerkezet az az *n*×*n*-es mátrix, ahol *n* a gráf csúcsainak száma, és minden csúcspárra megadja, hogy van-e közöttük él.. Az *A* = (*ai*,*j*) mátrix *G* gráf *szomszédsági mátrixa*, ahol *ai,j* 1, ha van él *i* és *j* csúcs között, különben 0. Irányítatlan gráfok esetén ez egy szimmetrikus mátrix lesz, amelynek hurokélek hiányában a diagonális elemei 0-k lesznek. Összeadva a mátrix *i*-edik oszlopát vagy sorát, az *i* ∈ *V*csúcs fokszámát kapjuk. A szomszédsági mátrix hátránya az *O*(*n*2) tárigénye, ugyanis az összes csúcspárhoz tárolunk egy számot. Ritka gráfok esetén nem hatékony, mert a kevés él mellett eltároljuk a nem szomszédos csúcspárokhoz tartozó 0-kat is. Előnye, hogy konstans időben kereshetjük, hogy (i,j) él megtalálható-e a gráfban, a mátrix i-edik sorának j-edik elemével.

Egy egyszerűbb opció az *éllista*, amely a gráf éleire illeszkedő végpontokat sorolja fel. Súlyozott gráfok esetén két végpont mellett egy harmadik adatot is tárol, az él súlya. Az éllista előnye, hogy kizárólag a szomszédos csúcsokat tartalmazza, ritka gráfok esetén a csúcspárok közötti élek hiányát nem kell eltárolni. Tárigénye *O*(*m*), ha m a gráf éleinek száma. Hátránya az élkeresés időigénye, ami *O*(*di*), ha *di* az *i* csúcs fokszáma.

*Szomszédsági listának* nevezzük azt a gráfreprezentációt, ahol a gráf minden pontjához tartozik egy lista, amely a pont szomszédait tartalmazza. Ha a gráf éleinek száma *m*, csúcsainak száma *n*, akkor a szomszédsági listák tárigénye *O*(*m*+*n*), ezért hatékony ritka gráfok esetén. Hátránya, hogy az élkereséshez legrosszabb esetben végig kell menni a gráf összes élén.

**2.3. Sztochasztikus mátrixok**

Egy *sztochasztikus mátrix* egy Markov-lánc átmeneteit leíró négyzetes mátrix. A gráf *R* *jobb oldali sztochasztikus mátrixa* egy diagonális mátrix, ahol a főátlóbeli elemek azt jelentik, hogy adott csúcsból mekkora valószínűséggel léphetünk egy másik csúcsra.Úgy kapjuk meg, hogy az *A* szomszédsági mátrixának minden i-edik sorát elosztjuk a *vi* csúcs fokszámával. Az *R* mátrix minden sorában lévő elemek összege 1.

A *T* *bal oldali sztochasztikus mátrix, átmenetmátrix* vagy *átmenetvalószínűség mátrix* az *R* mátrix transzponáltja. A *T* mátrix egyes oszlopaiban lévő elemeinek összege 1.

**2.4. Közösségek gráfokban**

A hálózatok közösségi szerkezetének vizsgálata során a korábbi munkák a közösségkeresést kétféle irányból közelítették meg. Az egyik irány egyszerre egy közösséget azonosít, a hálózat egy sűrű részgráfját, így megengedi, hogy egy pont több közösséghez is tartozzon. Ezeket nevezzük *átfedő közösségeknek*. A másik a gráf pontjainak *klaszterezése* vagy *osztályozása,* ami a hálózat diszjunkt közösségekre való felosztását jelenti. A továbbiakban a közösségkeresés alatt a klaszterezést értjük, vagyis egy csúcs pontosan egy közösséghez tartozik.

A *hálózatot* egy *G* = (*V*, *E*) gráfként lehet megadni *V*={*v1*, *v2*, …, *vn*} csúcsok és *E*={*eij* | *vi*, *vj*∈*V* és *i*≠*j*} élek halmazával, ahol *n*-nel jelöljük a *csúcsok számát*, *m*-mel az *élek számát*.

A hálózatokban megfigyelhetők erősen interaktív ponthalmazok, amelyekben a csúcsok közös tulajdonságokat és kapcsolatokat teremtenek és osztanak meg egymással. Ilyen halmaz lesz a *közösség*, vagyis *C* ⊆ *V* csúcsok részhalmaza, amelyben a részhalmazon belüli csúcsok közötti élek sűrűsége nagyobb, mint ezt a részhalmazt a többivel összekötő élek sűrűsége.

A *közösségkeresési problémának nevezzük V* felosztását *C* = {*C1*, *C2*, …, *Ck*} diszjunkt közösségek halmazára. A hálózatok optimális közösségszerkezetét nevezzük *alapigazságnak*.

A *µ* *topological mixing paraméter* azt határozza meg, hogy az egyes csúcsok éleinek mekkora hányada kapcsolódik a közösségén kívülre. Tehát a csúcsok 1 – µ hányada kapcsolódik közösségen belüli élhez. A közösségszertkezettel rendelkező hálózatok fokszámai és a közösségméretek eloszlása hatványtörvény γ ás béta exponenssel. A β a fokszámsorozat mínusz exponense, a γ a közösségek méreteloszlásának mínusz exponense. A valós hálózatok γ és β exponenseinek tipikus értékei: 2 ≤ γ ≤ 3, 1 ≤ β ≤ 2.

**2.5. Modularitás**

A partícionálás minőségének mérésére több mérték létezik, az egyik legszélesebb körben használt és legismertebb a Newman által bevezetett *modularitás*.

Az ötlet azon alapul, hogy a véletlen hálózatok nem mutatnak közösségi szerkezetet. Vegyünk egy tetszőleges hálózatot és a hálózat egy tetszőleges felosztását nc közösségre. Legyen e egy nc x nc méretű mátrix, amelynek az eij eleme

Ahhoz hogy össze lehessen hasonlítani különböző méretű hálózatok modularitását, célszerű normalizálni a különbséget az tényezővel. Így a modularitás értéke -1 és 1 közé eső szám lesz. Ha *A* = (*ai,j* ) *G* szomszédsági mátrixa, *di* a *vi* ∈ *V* csúcs fokszáma, *m* az élek száma és *n* a csúcsok száma *G*-ben, akkor egy adott *C* felosztás *Q modularitásának* definíciója:

ahol *σ*(*i, j*) 1, ha *i* és *j* ugyanabban a közösségben vannak, különben 0.

Minden élnek a valószínűsége, hogy összeköti *i* és *j* pontokat. Azért kell 2-vel szorozni, mert az él két végpontjából bármelyik lehet az *i*. Az *i* és *j* közötti élek számának várható értéke tehát .

Az elemek alkotják *G* gráf *M* *modularitás mátrixát*.

Valójában egy adott felosztás modularitása a közösségeken belüli élek száma mínusz a közösségeken belüli élek számának várható értéke egy azonos fokszámeloszlású véletlen gráfban. Tehát azok a jó minőségű közösségszerkezetek, amelyeknek nagy a modularitása. Ha a talált szétosztás nem tartalmaz több közösségen belüli élt, mint ami egy véletlenszerű hálózatban várható lenne, akkor a modularitás 0. Pozitív értékű modularitás azt jelzi, hogy a hálózatnak van közösségszerkezete, Q=0,3 feletti értékek már erős közösségszerkezetre utalnak.

**3. Eljárások**

A hálózatok közösségszerkezetének felderítése a hálózattudomány és a gráfelmélet egyik alapvető témájává vált, és a tudomány széles területén alkalmazzák. A probléma iránti nagy érdéklődés miatt számos különféle algoritmus született a hálózati közösségek azonosítására. Ezek közül a szakdolgozatom egy lineáris programozási megközelítést és egy az utazó ügynök problémára alapuló módszert mutat be, és hasonlít össze.

**3.1 IP alapú**

**3.1.1 Modularitás maximalizálási probléma?**

Newman [15]-ben a modularitás maximalizálását javasolta a közösségkeresési probléma megoldására. A *modularitás maximalizálási probléma* célja a modularitást maximalizáló partícionálás megtalálása. Az ötlete abból ered, hogy ha a magas modularitás jó közösségszerkezetet jelent, akkor már a modularitást maximalizálja a az összes lehetséges partícionálásból, hogy megtalálja a legjobbat. Exponenciális idő lenne végig menni az összes lehetőségen, hogy megtaláljuk a maximális modularitást, ami a gyakorlatban megvalósíthatatlan 20-30 csúcsnál nagyobb méretű hálózatok esetében. Ennek elkerülésére a közelítő megoldást adó módszereket haszálhatjuk, mint a szimulált hűtés és a genetikus algoritmusok. Newman egy mohó algoritmusra alapuló eljárást javasolt.

A modularitás maximalizálási probléma NP-nehéz, azonban számos algoritmus létezik a megoldására, heurisztikák és egzakt módszerek is. []-ban egészértékű lineáris programozási modellként fejezték ki a modularitás maximalizálási problémát. Ezt a módszert implementáltuk.

**3.1.2 Egészértékű lineáris programozási modell leírása**

Adott egy G = (V,E) irányítatlan gráf. Legyen A = (ai,j ) G szomszédsági mátrixa, di az i-edik csúcs fokszáma, m az élek száma és n a csúcsok száma G-ben. Legyen xij a döntési változó minden csúcspárhoz. xij 0, ha i és j csúcsok ugyanahhoz a közösséghez tartoznak, különben 1.

Legyen Iall = {(i, j) ∈ V2| i < j}

qij = ai,j –didj/2m G modularitás mátrixa, elemei konstansok, ezért lesz a céfüggvény lineáris.

Cél a modularitás maximalizálása, ezért a célfüggvény:

max 1 m Σ (i,j)∈Iall qij (1 − xij ) (IP-M)

A feltételeknek garantálniuk kell, hogy ha i és j csúcsok ugyanabban a közösségben vannak, és j és k szintén ugyanabban a közösségben vannak, akkor i és k is:

xij + xjk − xik ≥ 0 ∀i < j < k

xij − xjk + xik ≥ 0 ∀i < j < k

− xij + xjk + xik ≥ 0 ∀i < j < k

xij ∈ {0, 1} ∀(i, j) ∈ Iall

Az egészértékű programok megoldása NP-nehéz.

**3.2. TSP alapú**

Az *utazó ügynök probléma (Travelling Salesman Problem, TSP)* az egyik legjobban kutatott probléma az optimalizálás területén, [1] – ben javasoltak a TSP-re alapuló közösségkereső algoritmust. Az utazó ügynök problémacélja, hogy adott városból indulva, adott N város halmazán találjuk egy minimális költségű utat úgy, hogy minden várost pontosan egyszer látogatunk meg, és visszatérünk a kezdő városba. Bármely két város közötti távolság ismert. Utazó ügynök problémaként modellezték a közösségkeresési problémát, egy csúcsot egy városként tekintve. Az ötlet abból ered, hogy mivel egy minimális költségű utat keresünk, az egymáshoz közeli városok általában klasztereződnek az útvonalon. A megoldásként kapott út néhány kisebb útra való szétdarabolásával kaphatjuk meg a hálózat közösségszerkezetét.

**3.2.1. PRF mátrix: PageRank Feature**

Az utazó ügynök problémában a városok közötti távolság előre meghatározott, ezért a közösségkeresési probléma modellezéséhez definiálni kell az egyes csúcspárok közötti távolságot. A távolság azt jelzi, hogy mekkora a valószínűsége, hogy a két csúcs egy közösséghez tartozik. Minél kisebb a távolság, annál nagyobb a valószínűsége, hogy azonos közösségbe kerültek.

A Google keresőmotorjának PageRank funkciója [2] alapján tervezték meg a PageRank Feature-t, amellyel ki tudjuk számítani a megfelelő távolságot. A PageRank egy weboldal rangsoroló algoritmus, amely a Web hivatkozáshálózatának felhasználásával rekurzívan számítja ki az oldalak centralitását. Egy weboldalnak akkor magas a PageRank értéke, ha sok más oldal mutat rá, vagy a rá mutató weboldalak magas PageRank-kel lettek minősítve.

Tegyük fel, hogy van egy *N* oldalból álló weboldal-hálózat, ahol a felhasználó az oldalakon lévő hiperhivatkozásokkal lép az egyik oldalról a másikra. Annak a valószínűségét, hogy t lépés után a pi oldalra érkezünk a következőképpen definiálták [1]-ben:

, ahol annak a valószínűsége, hogy *pj*-ből *pi*-be lépünk, és a *d*∈[0, 1] paraméter egy kiugró faktor. Erre azért van szükség, hogy a felhasználó ne ragadjon hivatkozás nélküli oldalon, és mert a felhasználó nem mindig hiperhivatkozással lép tovább, hanem egy random oldalon folytatja a keresést. Egy másik oldalt a felhasználó *d* valószínűséggel látogat meg hiperhivatkozás használatával, és 1-*d* valószínűséggel hiperhivatkozás nélkül. A PageRank kezdőértéke egy *N* oldalt tartalmazó hálózaton:

Számos iteráció után egy konkrét értékhez konvergál, ez lesz az oldalak PageRank értéke, amely valójában azt mutatja meg, hogy egy felhasználó mekkora valószínűséggel látogat meg egy weboldalt. A PageRank értékek a weboldalak valószínűségi eloszlását alkotják, az összegük egy lesz.

A TSP algoritmusok futtatásához, és ezzel hálózat szerkezetének meghatározásához azonban az egyes csúcspárok közötti távolságra van szükségünk, amelyet a *PRF mátrix* elemeiből tudunk kiszámítani. A közösség definíciója szerint a hálózatokban a közösségen belüli csúcsok közötti élek sűrűbbek, mint a közösségek között. Ha a hálózat *Ci* közösségének *vi*∈*V* csúcsából indul az ügynök, akkor nagyobb a valószínűsége, hogy *t* lépés után egy ugyanabban a közösségben lévő csúcsban áll meg, mint egy másik közösségben. A PageRank-kel ellentétben, PRF értéke nem csak attól függ, hogy hova érkezünk, számít az is, hogy melyik csúcsból indítjuk az utat, ugyanis a TSP problémában a kiindulási pont adott. Ahhoz, hogy megkapjuk a csúcspárok közötti távolságot, a PRF(*vi*) vektornak azt kell megmutatnia, hogy melyik csúcsban állunk meg a legnagyobb valószínűséggel, ha a kezdő csúcs egy meghatározott *vi* pont.

A PRF mátrix *i*-edik oszlopvektora a *vi* csúcshoz a *t* lépés utáni végpontok valószínűségi eloszlása, az alábbi képlettel definiálták:

, ahol , *j*=1, 2, …, *N* annak a valószínűsége, hogy a *vi* csúcsból indulunk, és a *vj* csúcsban állunk meg *t* lépés után. Ha *vi* a megadott kezdő csúcs, akkor *PRF*(*vi*) kezdőértéke (*PRF0*(*vi)* ) egy vektor, ahol az i-edik elem 1, minden más elem 0.

A PR alapján a PRF mátrixot a következőképpen lehet kiszámítani:

t = 2, 3, …

A *T* mátrix a hálózat valószínűségátmenet mátrixa. *T*[*i,j*] annak a valószínűsége, hogy *vi* csúcsból 1 lépéssel *vj* csúcsba lépünk.

Azt mondtuk, hogy a közeli csúcsok klasztereződnek, ezért a csúcsok távolságából szeretnénk megállapítani a közösségszerkezeteket. Mivel a *PRF* vektorok elemeit használjuk a távolságok kiszámítására, az azonos közösségben lévő csúcsok PRF vektorainak hasonlónak kellene lennie, és különböznie kellene különböző közösségben lévő csúcsokétól. A véletlen séta során *vi* csúcsból indulva hasonló valószínűséggel érkezünk vk (*k*=1, 2, …, *N)* csúcsba, mint ha a *vj* csúcsból indulnánk, hiszen azonos közösséghez tartoznak.

Ahogy *PR,* úgy *PRF* is egy konkrét értékhez konvergál, ezért, ha *t* elég nagy, akkor *vi* és *vj* csúcsok *PRF*t(*vi*) és *PRF*t(*vj*) értéke majdnem megegyezik, nem lenne alkalmas a közösségek megadására. Ha 0 lépés után állnánk meg, akkor az értékek teljesen különböznének minden csúcs külön közösségbe kerülne, végtelen lépés esetén az összes ugyanannyi lenne, egy közösségben helyezkedne el az összes csúcs. [1]-ben a *t* paraméter értékének a 6-ot választották, ezzel az összes csúcsot lefedve a hálózatban, de még a *PRF* vektorok nem közelítenek túlságosan egymáshoz.

**3.2.2. PRD mátrix: PageRank Distance**

A PRF mátrix kiszámítása után minden csúcs leképezhető egy *N* dimenziós Hilbert térbe, ahol a pontok koordinátáit a csúcs PRF vektora adja meg. Bármely két csúcs távolsága az euklideszi távolsággal kiszámolható, amelynek a definíciója az alábbi:

Az összes csúcspár kiszámításának az időigénye *O*(*N*3). Felfedezték, hogy a hálózatok RWF mátrixának elemei között sok hasonló és kicsi pont van. A számítási költség csökkentésére a PRD mátrix kiszámítása során csak a kiugró értéket vették figyelembe, a triviális pontokat 0-val helyettesítették, így csak a kiugró értékekhez tartozó csúcsok távolságait kell kiszámolni. A peak pontok halmazát a következőképpen definiálták:

Vagyis azon csúcsok halmaza, ahol a PRF érték a legnagyobb az adott csúcs (*vi*) PRF vektorában. ha i-ből indulunk az a legvalószínűbb, hogy j-ben állunk meg

Egy *N* ponttal rendelkező hálózatban a PRF vektorok száma *N*, tehát legfeljebb *N* eleme lesz a *PP* halmaznak. Előfordulhat, hogy több PRF vektornak is ugyanabban a pontban van a legnagyobb értéke, főleg azoknál a csúcsoknál, amelyek ugyanahhoz a közösséghez tartoznak, ezért általában a PP elemeinek száma jóval kisebb lesz N-nél. A PP pontok halmazába általában a komplex hálózat azon néhány csúcsa kerül, amelyeknek nagy a fokszáma, és nagy valószínűséggel ezekben a csúcsokban fogunk megállni a véletlen séta végén.

Ezek alapján a PRD-t a következőképpen definiálták:

A PP halmazzal lecsökkentették a PRF mátrix dimenzióját a halmaz elemeinek számára. Ez a szám különböző hálózatoknál eltér, azonban gyakran a közösségek számának nagyságrendjében van.

**3.2.3. TSP solver**

A közösségkeresési probléma célja, hogy találjunk modulokat a hálózatokban, ahol a modulon belüli csúcsok sűrűbben vannak összekötve, mint a többi modulokhoz tartozók. Minél kisebb a PRD, vagyis minél kisebb a távolság két csúcs között, annál nagyobb a valószínűsége, hogy ugyanabban a közösségben vannak.

Miután megkaptuk a PRD mátrixot a távolságokkal, a közösségek azonosításához először használnunk kell egy TSP megoldót, amiből megkapunk egy minimális költségű körutat, amely az összes pontot bejárja. Ehhez bármely TSP algoritmust használhatjuk. Annak ellenére, hogy az utazó ügynök probléma NP-nehéz, számos heurisztika és egzakt módszer létezik a megoldására.

A python-tsp[] egy Python könyvtár a TSP probléma megoldásához. Egzakt és heurisztikus metódusai egy távolság mátrixot várnak paraméterként. A távolság mátrix egy nxn méretű numpy tömb, n a hálózat csúcsainak száma. Egy minimális költségű úttal és az út hosszával térnek vissza. A kapott út egy kör, azonban a függvény a csúcsok listájaként adja vissza egész számokkal jelölve a csúcsokat 0-tól n-ig.

A heurisztikák esetén a megoldás futásról futásra változhat, és nem garantálja az optimalitást, viszont nagyobb hálózatok esetén sokkal hatékonyabb időben. Teszt hálózatainkon az egzakt megközelítés túl sokáig futna, ezért a teszteléshez két heurisztikát választottam.

A helyi kereső eljárás (solve\_tsp\_local\_search) az indulóponthoz tartozó helyi minimumot találja meg. Adott egy kezdő kör, ha nem adjuk meg a függvény második (opcionális) paramétereként, akkor az algoritmus egy véletlen úttal indul. Új szomszédokat hoz létre, addig amíg már nem talál jobb szomszédokat az aktuálisnál, ez lesz a lokális optimum. A lokális optimum nem biztos, hogy meg fog egyezni a globális optimummal, főleg nagy hálózatok esetén.

A szimulált hűtés is egy lokális kereső algoritmus, azonban lassabb, és kisebb az esélye, hogy helyi minimumban ragad. Mint a solve\_tsp\_local\_search, a solve\_tsp\_simulated\_annealing eljárás is véletlen úttal kezd, ha a x0 paramétert nem adjuk meg.

**3.2.4. Automatikus küszöbérték alapú vágás**

Miután megvan az optimális út, már csak az kérdés, hogyan alakítjuk át az utat egy közösség szerkezetté. Meg kell határoznunk, hogy hol vágjuk el az útvonalat kisebb útszakaszokra. Ezen szakaszok pontjai alkotják a közösségeket. Ha a hálózatnak van egy jó közösségszerkezete, akkor a távolság két azonos közösségben lévő csúcs között sokkal kisebb, mint a távolság két különböző közösségben lévő csúcs között. Először a nagyobb távolságok mentén kellene felosztani a pontokat.

Hierarchikus struktúrát kapunk, ha csökkenő sorrendben végezzük a vágást. A legelső vágás a kört vágja el, a többi vágás az adott utat osztja két kisebb útszakaszra. Az utolsó vágás után minden út egy közösségnek felel meg.

Ahhoz, hogy ne vágjunk közösségen belüli távolságot, szükségünk van egy határértékre, ami megmondja, hogy mekkora távolságnál fejezzük be a vágást. Azért, hogy az iterációk száma ne legyen nagyobb, mint n/2, []-ben a a távolságok átlagának és szórásának összegét javasolták küszöbértéknek.

ahol Di az optimális út i-edik és az i+1-edik csúcsának PRD értéke.

**4. Tesztelés**

Ebben a fejezetben az algoritmusok eredményeit hasonlítjuk össze NMI pontosság és futásidő szerint. A lineáris programozási modell esetében a modularitás értékét is figyelembe vesszük. Az algoritmusok teljesítményeit valós világbeli és számítógéppel generált hálózatokon értékeljük ki. Az eljárások minőségének méréséhez célszerű olyan hálózatokon tesztelni, amelyek rendelkeznek közösségszerkezettel, hogy az algoritmusok megtalálhassák. Szükséges ismerni a teszt hálózatok közösségeit. Ezeket fogjuk alapigazság vagy optimális közösségeknek nevezni. A teszteléshez kis méretű gráfokat használunk az algoritmusok komplexitása miatt.

A tesztek egy Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz 1.19 GHz processzorral, 8,00 GB memóriával ellátott számítógépen, Windows 10 operációs rendszeren lettek elvégezve.

**4.1. NMI pontosság**

A klaszterezések kiértékeléséhez az egyik jól ismert teljesítmény mérő az *NMI (normalized mutual information)*. Az NMI az optimális közösségek és az algoritmus által talált közösségek hasonlóságát minősíti. Adott G hálózat optimális közösségeinek halmaza legyen C(A), és az adott algoritmusból kapott közösségeinek halmaza legyen C(B).

ahol n a hálózat pontjainak száma, CA az alapigazság közösségek száma, CB a talált közösségek száma, nij azon i valós alapigazság közösségben lévő csúcsok száma, amelyek megtalálhatók a j talált közösségben is, ni az i alapigazság közösségben lévő csúcsok száma, nj a j talált közösségben lévő csúcsok száma.

Az NMI értéke 0 és 1 közötti valós szám, minél jobban hasonlít az adott algoritmus által visszaadott közösségi hozzárendelés az alapigazságra, annál nagyobb. Abban az esetben, ha a talált közösségek azonosak az alapigazsággal, az NMI egy, ha teljesen különböznek, akkor nulla.

A pontosság mérésére elkészítettem Python nyelven a saját implementációmat nmi.py néven. Az nmi(A,B) függvény két paramétert vár, az egyik az alapigazságot, a másik a talált közösségeket tartalmazó DAT fájl. Az input fájlok a közösségszerkezetet csúcs és a hozzárendelt közösség párok listájaként írják le. A program beolvassa a fájlokat, kiszámítja a talált közösségszerkezet NMI értékét, és ezt az értéket adja vissza.

Az nmi érték kiszámításához a kapott közösségszerkezetet kiíratjuk egy DAT fájlba a TSP alapú eljárás esetén a get\_tsp\_communities(membership, cuts, file), az IP alapú eljárás esetén a def get\_ip\_communities(matrix, communities) függvénnyel. A fájl a hálózat pontjait tartalmazza soronként növekvő sorrendben 1-től kezdve egész számokkal jelölve. Minden pont mellé a hozzárendelt közösség száma kerül, a közösségeket egész számokkal jelöljük 1-től kezdve.

**4.2. Valós világbeli gráfok**

A közösségkereső algoritmusok tesztelésére léteznek kis méretű valós világbeli standard teszt gráfok, amelyeknek korábbi tanulmányokból ismerjük a közösségszerkezetét. A teszteléshez az 1. táblázatban lévő hálózatokat választottam. A teszt hálózatok megtalálhatók Mark Newman oldalán [] GML formátumban. A gráfokat a NetworkX csomag és a saját függvényeim segítségével alakítottam át az algoritmusoknak megfelelő formátumra. A teszteléshez szükségem volt a gráfok éllistájára, szomszédsági mátrixára, fokszámsorozatára, alapigazság közösségszerkezetére. Az optimális közösségszerkezeteket a hálózat forrásaként hivatkozott cikkekből vagy a GML fájlból állítottam elő.

|  |  |  |
| --- | --- | --- |
| Hálózat | *n* | *m* |
| Zachary's karate club | 34 | 78 |
| Dolphin social network | 62 | 159 |
| Les Miserables | 77 | 254 |
| Books about U.S. politics | 105 | 441 |
| College Football | 115 | 613 |

1.táblázat. Való világbeli teszt hálózatok. Az n a csúcsok száma, az m az élek száma.

Zachary karate klub [karate] hálózata egy amerikai egyetem karate klubjának barátságait jelöli. Tudjuk, hogy a 34 tagból álló klub egy konfliktus következtében két külön klubra bomlott, és az algoritmusok által talált közösségszerkezetet ehhez tudjuk hasonlítani. Az 1. ábrán a az 1-es pont a klub oktatója, a 34-es pont a klub adminisztrátora.

1.ábra. A Zachary karate klubja gráffal ábrázolva a klub szétválása után.

A delfin szociális hálózat a Doubtful Sound palackorrú delfinjei közötti gyakori társulásokat írja le. []-ben 62 delfint figyeltek meg.

A Les Miserables hálózat csúcsai a Nyomorultak című regény szereplői. Ha két szereplőt összeköt él, az azt jelenti, hogy a szereplők megjelennek ugyanabban a fejezetben.

A College Football gráf[fball] amerikai futball mérkőzéseket reprezentál, ahol a csapatok a gráf csúcsainak felelnek meg, és két csapat között akkor helyezkedik el él, ha megmérkőztek egymással a szezonban. A csapatok szét lettek osztva 8-12 csapatból álló konferenciákra. Konferencián belül gyakoribbak voltak a mérkőzések, mint konferenciák között, egy csapat átlagosan 7 meccset játszott a konferenciáján belüli és 4 meccset a konferenciáján kívüli csapatokkal. A 12 konferencia lesz az alapigazság, amit az algoritmusoknak meg kellene találniuk. A GML fájl value attribútum tartalmazza, hogy melyik csapat melyik konferenciához tartozik.

Books about U.S. politics [books] hálózat pontjai a az Amazon.com-ról vásárolt amerikai politikáról szóló könyveket reprezentálják, az élek a gyakran együtt vásárolt könyveket kötik össze. Newman szétosztotta a könyveket a leírásuk és az Amazonon közzétett vélemények alapján liberális és konzervatív könyvek osztályára. A könyvek egy részéről nem tudta egyértelműen eldönteni, melyik ideológiához tartoznak. A GML fájl value értékében van megadva, hogy melyik könyvet melyik ideológiához sorolta.

**4.3. Generált gráfok**

A valós világbeli hálózatainkat kiegészítjük néhány nagyobb méretű számítógéppel generált véletlenszerű hálózattal. Ezeknek az előnye, hogy beépített közösségszerkezetüket ismerjük, és paraméterekkel szabályozhatjuk.

A leggyakrabban viszonyítási alapként használt gráfok a Girvan és Newman gráfok, amelyekben a 128 pont 4 egyforma méretű, 32 csúcsból álló közösségre oszlik. Azonban a valós hálózatokat a csúcsok fokszámának heterogén eloszlása jellemzi, és a közösségek mérete sem egyforma. A [2]-ban leírt algoritmus erre a problémára nyújt megoldást.

A mesterséges hálózatainkat az algoritmusuk implemetációjával[2] állítjuk elő, amely véletlenszerű hálózatokat generál beépített közösségszerkezettel a megadott paraméterektől függően. A program három fájlt készít, ezekből a hálózat éllistájára és a közösségszerkezetre van szükségünk. A network.dat a generált hálózat éllistáját tartalmazza soronként, a csúcsokat egész számokkal jelöli 1-től kezdve, növekvő sorrendben úgy, hogy minden él kétszer szerepel. A community.dat-ban a hálózat csúcsainak listája szerepel, a csúcsokhoz hozzárendelve az őket tartalmazó közösséget. A közösségeket is egész számokkal jelöli, 1-től indítva a számozást.

A hálózatok generálásához a -N, -k, -maxk, -mu paramétereket kötelező megadni, ahol a –N a hálózat csúcsainak száma, -k az átlagos fokszám, -maxk a maximális fokszám, -mu a mixing paraméter, -minc a közösségek minimális mérete, -maxc a közösségek maximális mérete, t1 a β exponens, t2 paraméter a γ exponens.

Egy megbízható teszt hálózatban változó a közösségek mérete és a csúcsok átlagos fokszáma. Tehát a k = 15, maxk = 50, minc = 20, maxc = 40 értékeket választottam. A mu mixing paraméter 0,1 és 0,5 értékeket kapott. mű = 0,1 értékkel a hálózatnak erős a közösségszerkezete, és 0,5 alatt még a csúcsoknak több szomszédja van a saját közösségén belül, mint közösségen kívüli szomszédja. Tehát egy mu = 0.5 értékű hálózatban a vizsgált algoritmus nehezebben találja meg az optimális közösségeket. A t1 = 2 és t2 = 1 alapértelmezett értékeken az esetek felében változtattam, ahol a t1-et 3-ra, a t2-t 2-re állítottam.

A program alapvetően átfedő közösségek felderítésének tesztelésére készült, azonban az átfedő csúcsok számának (–on) és az átfedő csúcsok tagságai számának (–om) megadása opcionális. Mivel az eljárásokkal diszjunkt közösségeket keresünk, a –on és –om paramétereket meghagyjuk az alapértelmezett 0 értékükön.

**4.4. IP alapú eljárás**

Az IP alapú modell-t AMPL-lel (A Mathematical Programming Language) [] implementáltam. Az AMPL egy magas szintű matematikai modellező nyelv, amellyel a legtöbb kereskedelmi és nyílt forráskódú megoldót tudjuk használni a modellek megoldására. Az AMPL lefordítja a felhasználó által definiált modellt a megoldó számára érthető alacsony szintű formára, visszaolvassa a megoldást a megoldótól, és értelmezi a magasabb szintű modellen belül. A lineáris program megoldására a Gurobi megoldó 11.0.1 verzióját használtam.

A Gurobi egy nagy teljesítményű matematikai programozási megoldó, amely a lineáris programozásra (LP), a kvadratikus programozásra (QP), valamint a kvadratikusan korlátozott programozásra (QCP, SOCP) és ezek vegyes egészértékű változataira (MIP, MIQP, MIQCP, MISOCP) specializálódott.

Az elkészített modellt az ip.mod fájl tartalmazza, amelyet az ip.run fájl segítségével futtattam. Minden teszt gráf esetében egy [gráf neve]\_ip.dat fájl tartalmazza a modell megoldásához szükséges adatokat, és egy [gráf neve]\_x.dat nevű fáljba írja ki a kapott modularitás és az xij értékeit.

A futási idő méréséhez AMPL-ben a „display \_ampl\_elapsed\_time +\_solve\_elapsed\_time;” parancsot használtam a futtató scriptben. Az „\_ampl\_elapsed\_time” az AMPL folyamat által eltöltött idő másodpercben, a megoldó idejét nem számítva, a „\_solve\_elapsed\_time” a "solve" meghívása és a megoldás visszakapása között eltelt időt adja vissza másodpercben.

|  |  |  |  |
| --- | --- | --- | --- |
| hálózat | *n* | *Q* | *c* |
| Zachary's karate club | 34 | 0,234796 | 4 |
| Dolphin social network | 62 | 0,274959 | 5 |
| Les Miserables | 77 | 0,291869 | 10 |
| Books about U.S. politics | 105 | 0,270384 | 5 |
| College Football | 115 | 0,306662 | 6 |

2.táblázat. Valós világbeli hálózatok. Az n a csúcsok száma, a Q az élek száma, a c a közösségek száma.

2.ábra. A karate klub IP algoritmus által talált közösségszerkezete. A csúcsok színezésével különítjük el a közösségeket.

**4.5. TSP alapú eljárás**

A TSP alapú algoritmus implementációját a Python 3.9.0 verziójával készítettem el. Az algoritmus kiszámítja az adott gráf összes csúcspárjának PRF értékét a PRF\_matrix(G, t) függvénnyel, majd kiválasztja a PP halmaz elemeit, és kiszámítja a csúcspárok PRD távolságát egy mátrixba a PRD\_matrix(G, t) függvénnyel. G paraméter az adott gráf szomszédsági mátrixa, t paraméter a lépések száma. A NumPy csomag 1.62.2-es, a NetworkX csomag 3.2.1-es verzóját használtam. A következő lépésben meghatároztuk az utat a python-tsp csomag 0.4.0-s verziójának a solve\_tsp\_simulated\_annealing(distance\_matrix) és a solve\_tsp\_local\_search(distance\_matrix). A függvényeknek egyetlen paramétert adunk, a távolság mátrixot. Ezután a calculate\_threshold(PRD, tour) függvény kiszámolja a küszöbértéket, ahol megállítjuk a vágást. Paraméterei a távolság mátrix és a kapott út, a küszöbértékkel tér vissza. Végül a split\_path(PRD, tour) függvénnyel feldaraboljuk az utat, és minden csúcshoz meghatározzuk, hogy melyik közösséghez tartozik. Paraméterként a távolság mátrixot és a kapott utat várja, a csúcsokat és a csúcsot tartalmazó értékeket adja vissza (csúcs: közösség) formában.

3.ábra. RWD értékek a TSP szerinti sorrendben egy vágással

A futási időt Pythonban a timeit modul[] használatával mértem. A timeit kisebb kódrészletek futási idejének mérésére alkalmas, másodpercben adja vissza időt. Az eltelt időt a távolság mátrix kiszámításától a közösségek meghatározásáig mértem.

**4.5. Eredmények összehasonlítása**

NMI

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Hálózat | *n* | *c* | IP | IP-*c* | SA | SA-*c* | LS | LS-*c* |
| Karate | 34 | 2 | 0,5878 | 4 | 0,3057 | 4 | 0,2401 | 4 |
| Dolphin | 62 | - | - | 5 | - | 9 | - | 9 |
| Les Miserables | 77 | - | - | 6 | - | 4 | - | 4 |
| Books | 105 | 3 | 0,5602 | 5 | 0,5611 | 14 | 0,5967 | 15 |
| College Football | 115 | 12 | 0,8800 | 10 | 0,9457 | 18 | 0,9354 | 16 |

3.táblázat. NMI értékek

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Hálózat | *n* | *γ* | *β* | *k* | *µ* | *c* | IP | IP-*c* | *Q* | SA | SA-*c* | LS | LS-*c* |
| 1 | 100 | 2 | 1 | 10 | 0.1 | 3 |  |  |  |  |  |  |  |
| 2 | 100 | 2 | 1 | 10 | 0.5 | 3 |  |  |  |  |  |  |  |
| 3 | 100 | 3 | 2 | 10 | 0.1 | 3 |  |  |  |  |  |  |  |
| 4 | 100 | 3 | 2 | 10 | 0.5 | 3 |  |  |  |  |  |  |  |
| 5 | 100 | 2 | 1 | 10 | 0.5 |  |  |  |  |  |  |  |  |
| 6 | 100 | 2 | 1 | 10 | 0.5 |  |  |  |  |  |  |  |  |

4.táblázat. NMI értékek

Futási idő

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hálózat | *n* | IP | TSP-SA | TSP-LS |
| Karate | 34 | 0,328 | 0,876 | 0,032 |
| Dolphin | 62 | 28,281 | 1,487 | 0,274 |
| Les Miserables | 77 | 6,078 | 2,038 | 0,578 |
| Books | 105 | 53,625 | 4,895 | 1,065 |
| College Football | 115 | 80,766 | 5,628 | 1,680 |

5.táblázat. Egészértékű modell, TSP szimulált hűtéssel és TSP lokási kereséssel futási ideje valós világbeli hálózatokon.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Hálózat | *n* | *γ* | *β* | *k* | *µ* | IP | *Q* | TSP-SA | TSP-LS |
| 1 | 100 | 2 | 1 | 10 | 0.1 | 57,828 |  |  |  |
| 2 | 100 | 2 | 1 | 10 | 0.5 |  |  |  |  |
| 3 | 100 | 3 | 2 | 10 | 0.1 |  |  |  |  |
| 4 | 100 | 3 | 2 | 10 | 0.5 |  |  |  |  |
| 5 | 100 | 2 | 1 | 15 | 0.5 |  |  |  |  |
| 6 | 100 | 2 | 1 | 15 | 0.5 |  |  |  |  |

6.táblázat. IP, TSP-SA és TSP-LS futási ideje a mesterséges hálózatokon.

**5. Összefoglalás**

**Irodalomjegyzék**

[] FORTUNATO, S. Community detection in graphs. Physics Reports 486, 75-174 (2010)

[] Newman, M. E. J. The structure and function of complex networks. SIAM Review 45, 167-256 (2003)

<https://www.inf.u-szeged.hu/~london/Halozatok/halozat5.pdf>

<https://www.inf.u-szeged.hu/~london/Halozatok/halozat1.pdf>

<https://www.math.u-szeged.hu/~katai/diszmat2/eavaz/3grafelmelet.pdf>

<https://www.khanacademy.org/computing/computer-science/algorithms/graph-representation/a/representing-graphs>

<https://www.inf.u-szeged.hu/~pluhar/oktatas/grafalg.pdf>

<https://www.inf.szte.hu/~rfarkas/Alga17/9_ElemiGrafalgoritmusok.ppt>

<https://www.cs.fsu.edu/~burmeste/slideshow/chapter23/23-1.html>

A hálózatok tudománya Barabási Albert-László

tsp

[1] Z. Jiang, J. Liu, S. Wang, Traveling salesman problems with PageRank Distance on complex networks reveal community structure, Physica A (2016)

[2] S. Brin and L. Page, “The anatomy of a large-scale hypertextual Web search engine,” Computer Networks and ISDN Systems, 30, 107-117, 1998.

<https://github.com/fillipe-gsm/python-tsp>

modzularitás

Modularity and community structure in networks

Finding community structure in networks using the eigenvectors of matrices nem kell duplicate

ip modellhez

[] ARMAN FERDOWSI, ALIREZA KHANTEYMOOR Discovering Communities in Networks: A Linear Programming Approach Using Max-Min Modularity

[] G. Agarwal and D. Kempe, Modularity-maximizing graph communities via mathematical programming

[] D. Aloise, S. Cafieri, G. Caporossi, P. Hansen, S. Perron, and L. Liberti, “Column generation algorithms for exact modularity maximization in networks,” Physical Review E, vol. 82, no. 4, p. 046112, 2010

[] R. Fourer and D. Gay, The AMPL Book. Pacific Grove: Duxbury Press, 2002

NMI-hez

[] L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas, “Comparing community structure identification,” Journal of Statistical Mechanics: Theory and Experiment, vol. 2005, no. 09, p. P09008, 2005.

teszteléshez:

[]Community structure in social and biological networks

[]Directed, weighted and overlapping benchmark graphs for community detection algorithms

[]Benchmark graphs for testing community detection algorithms

[] <https://www-personal.umich.edu/~mejn/netdata/>

[karate] W. W. Zachary, An information flow model for conflict and fission in small groups, Journal of Anthropological Research 33, 452-473 (1977)

[dolphin] D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, and S. M. Dawson, Behavioral Ecology and Sociobiology 54, 396-405 (2003)

[Les Miserables] D. E. Knuth, The Stanford GraphBase: A Platform for Combinatorial Computing, Addison-Wesley, Reading, MA (1993)

[football] M. Girvan and M. E. J. Newman, Proc. Natl. Acad. Sci. USA 99, 7821-7826 (2002)

[books] M. Newman, Proc. Natl. Acad. Sci. USA 103, 8577