Szegedi Tudományegyetem

Informatikai Intézet

SZAKDOLGOZAT

Juhász Dóra

2024

Szegedi Tudományegyetem
Informatikai Intézet

Algoritmusok vizsgálata hálózati közösségek meghatározására

Szakdolgozat

Készítette:
Juhász Dóra
gazdaságinformatikus BSc
szakos hallgató

Témavezető:
Dr. Vinkó Tamás
egyetemi docens

Szeged
2024

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:

Hálózatok közösségszerkezetének azonosítása az utazó ügynök probléma és a modularitás maximalizálási probléma megoldásával. A két ismert algoritmus implementációja és tesztelése valós világbeli és mesterséges hálózatokkal. Az eredmények összehasonlítása futásidő és NMI pontosság szerint.

A megoldási mód:

- A témához kapcsolódó szakirodalom megismerése és feldolgozása.
- A modularitás maximalizálási probléma modellezése AMPL nyelven és az egészértékű lineáris program megoldása.
- A TSP probléma megoldásához szükséges távolság mátrix kiszámítása a PageRank Feature alapján, majd a TSP probléma megoldása két elérhető heurisztikus módszerrel, és a kapott út feldarabolása közösségekre.
- Szoftver által generált standard teszt hálózatok előállítása.
- A Normalized Mutual Information implementálása. Az ismert közösségek összehasonlítása az algoritmusok által meghatározott közösségekkel.

Alkalmazott eszközök, módszerek:

Az IP alapú algoritmus megvalósításához és teszteléséhez az AMPL-t és a Gurobi megoldót használtam. A távolság mátrix kiszámítását, a kapott út feldarabolását, az NMI megvalósítását és az adatfájlok átalakítását Python nyelven oldottam meg. Ezekhez főként a NetworkX, a python_tsp könyvtárak függvényeit alkalmaztam. A Python nyelven írt kódot a PyCharm fejlesztői környezetben, az AMPL modellt az AMPL IDE fejlesztői környezetben készítettem el. A hálózatok a Gephi 0.10.1-es verziójával ábrázoltam.

Elért eredmények:

A teszt hálózatokon közösségi szerkezet felderítése. A két algoritmus tesztelése során kapott eredmények összehasonlítása és kiértékelése.

Kulcsszavak:

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

Tartalomjegyzék

Bevezetés

A világban megtalálható számos rendszert reprezentálhatunk hálózattal. A hálózatok csúcsokból és a csúcsokat összekötő élekből állnak. Például a szociális hálózatokban az emberek csúcsoknak, a kapcsolataik éleknek feleltethetők meg. Weboldalak és tudományos cikkek is hálózatot alkotnak, ahol az élek a hivatkozásokat jelölik. A biológiában is találunk példát hálózatokra mint a neurális hálózatok, az anyagcsere hálózatok és a táplálékláncc.

A gráfokban felfedeztek olyan részgráfokat, amelyek csúcsai között sűrűbben helyezkednek élek, és kevesebb éllel kapcsolódnak más részgráfok csúcsaihoz, ezeket nevezzük közösségeknek. Ahhoz hogy elemezni tudjuk a hálózatokat, fontos a közösségszerkezetük felderítése. A közösségeken belüli csúcsok általában valamilyen közös tulajdonsággal rendelkeznek. Például közösségi hálózat esetén ez lehet közös érdeklődés, lakóhely, iskola, a weboldalakon közös téma, biológiai hálózatokban közös szerep. Ha egy hálózatot nincs értelme egészében elemezni, a közösségek azonosításával külön is vizsgálhatjuk a közösségeket. Vagy például túl nagy és komplex hálózatok csúcsait helyettesíthetjük úgy, hogy a közösségeiket csúcsként ábrázoljuk.

A közösségkeresési probléma a hálózattudomány és a gráfelmélet egyik alapvető témájává vált, és a tudomány széles területén alkalmazzák, például az informatikában, a fizikában, a biológiában, a gazdaságtudományban és a szociológiában. A közösségek detektálása nehéz feladat, de probléma iránti nagy érdeklődés miatt számos különféle algoritmust fejlesztettek a megoldására. A dolgozatomban ezek közül kettőt vizsgálok.

Az egyik módszer a modularitás maximalizálása lineáris programozással. Mivel a modularitás az azonosított közösségszerkezetet minősíti, érdemes a maximális modularitású közösségszerkezetet keresni. A modularitás függvény lineáris, ezért egy egészértékű lineáris programozási modell megoldása alkalmas erre. A másik módszer az utazó ügynök problémát használja úgy, hogy a hálózat csúcsai a városoknak felelnek meg, és a gráf két csúcsa közötti távolságok lesznek a városok közötti távolságok. Arra számítunk, hogy az optimális útvonal az egy közösségbe sorolt csúcsokat egymás után járja be, ezért ha a kiugróan nagy távolságok mentén feldaraboljuk, megkaphatjuk a közösségeket. Az utazó ügynök probléma megoldására két heurisztikát alkalmaztam, egy lokális kereső és egy szimulált hűtésen alapuló megoldót. A két eljárást való világbeli és különféle számítógéppel generált gráfokon tesztelem, amelyeknek ismert a közösségszerkezete. Az algoritmusok eredményeit futási idő és a NMI pontosság szempontjából hasonlítom össze.

1. Alapfogalmak

Ez a fejezet a szakdolgozatban használt kifejezések definícióit tartalmazza. A szakirodalomban a hálózat és gráf kifejezéseket általában szinonimaként használják, az utóbbit inkább a matematikában.

1.1. Gráfelméleti alapfogalmak

Egy $G = (V, E)$ *gráf* $V = \{1, 2, \ldots, n\}$ *csúcsok* halmazából és a csúcsokat összekötő $E \subseteq V \times V$ *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, $(i, j) \in V$ 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 $|E| \sim |V|^2$, 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, $|E| \sim |V|$.

Irányítatlan gráf $i \in V$ csúcsának *fokszáma* a csúcsból induló élek száma, d_i-vel jelöljük. A gráf *fokszámsorozata* a gráf csúcsainak fokszámaiból álló lista, $(k_1, k_2, \ldots, k_n)$. Az i és a j csúcs *szomszédok* vagy *szomszédosak*, ha összeköti őket él.

Egy $G' = (V', E')$ gráf $G = (V, E)$ gráf *részgráfja*, ha $V' \subseteq V$ és $E' \subseteq E$. *Vágásnak* nevezzük V csúcshalmaz felosztását két (S és $V - S$) halmazra.

G gráfban $v_0, e_1, v_1, \ldots, v_{k-1}, e_k, v_k$ sorozatot *sétának* nevezünk, ha $v_0, v_1, \ldots, v_k$ G csúcsainak sorozata, $v_1, \ldots, v_k$ G éleinek sorozata, és $e_i \in E$ két végpontja v_i-1 és v_i minden $i \in \{1, \ldots, k\}$ 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.

1.2. 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 $n \times n$-es mátrix, ahol n a gráf csúcsainak száma, és minden csúcspárra megadja, hogy van-e közöttük él.. Az $A = (a_{i,j})$ mátrix G gráf *szomszédsági mátrixa*, ahol $a_{i,j}$ 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 $i \in V$ 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 $O(d_i)$, ha di az i csúcs fokszáma.

Szomszédsági listának nevezzük azt a gráfreprezentációt, ahol a gráf minden pontjához tartozik egy lista, amely a pont szomszédjait 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.

1.3. 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 v_i 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.

1.4. 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 $V=\{v_1, v_2, \ldots, v_n\}$ csúcsok és $E = \{e_{ij} \mid v_i, v_j \in V \text{ és } I \neq j\}$ 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 $C \subseteq V$ 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 $C = \{C_1, C_2, \ldots, C_k\}$ diszjunkt közösségek halmazára. Azt szeretnénk meghatározni, hogy létezik-e a csúcsoknak természetes felosztása közösségekre, úgy, hogy a közösségek bármekkora méretűek lehetnek. A hálózatok ismert közösségszerkezetét nevezzük *alapigazságnak*.

A μ *mixing paraméter* azt határozza meg, hogy az egyes csúcsok éleinek mekkora hányada kapcsolódik a közösségén kívülre. Tehát a csúcsok $1 - \mu$ hányada kapcsolódik közösségen belüli élhez. Egy hálózat általában sok alacsony fokszámú csúcsot és kevés magas fokszámú csúcsot tartalmaz, és a közösségszerkezettel rendelkező hálózatokban a közösségméretek eloszlása is hatványtörvény követ. A β a *fokszámsorozat mínusz exponense*, a γ a *közösségek méreteloszlásának mínusz exponense*. A valós hálózatok γ és β exponenseinek tipikus értékei: $2 \leq \gamma \leq 3$, $1 \leq \beta \leq 2$.

2. Eljárások

Ebben a fejezetben egy lineáris programozási megközelítést és az utazó ügynök problémára alapuló módszert mutatok be.

2.1. IP alapú

2.1.1. 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 G hálózatot és a hálózat egy tetszőleges felosztását k közösségre. Legyen e egy k x k méretű mátrix, amelynek az e_{ij} eleme a hálózat azon éleinek hányada, amely az i közösségben lévő csúcsokat a j közösségben lévő csúcsokkal köti össze. Az e mátrix főátlójában lévő elemeinek összege lesz a hálózat azon éleinek hányada, amelyek az ugyanabba a közösségbe tartozó csúcsokat kötik össze. Ez az összeg jól minősíti a felosztást, minél nagyobb az értéke, annál jobb az azonosított közösségszerkezet. Önmagában viszont nem elég, mert ha egyetlen közösségbe helyeznénk a hálózat összes csúcsát, akkor elérnénk a maximális 1 értéket, de nem kapnánk meg a közösségszerkezetet. Legyen $b_i = \sum_j e_{ij}$ az e mátrix i-edik sorának összege, amely azon élek hányada, amelyek egy i közösségben lévő csúcshoz kapcsolódnak. Egy hálózatban a közösségek figyelembe vétele nélkül $e_{ij} = b_i b_j$, így a modularitás:

$$Q = \sum_i (e_{ii} - b_i^2) \tag{2.1}$$

Ha $A = (a_{i,j})$ G szomszédsági mátrixa, d_i a $v_i \in V$ csúcs fokszáma, m az élek száma és n a csúcsok száma G-ben, akkor egy adott C felosztás Q *modularitását* definiálhatjuk a következőképpen is:

$$Q(C) = \frac{1}{2m} \sum_{i,j \in V} \left[a_{i,j} - \frac{d_i d_j}{2m} \right] \delta(i,j) , \tag{2.2}$$

ahol $\delta(i, j)$ 1, ha i és j ugyanabban a közösségben vannak, különben 0. $\frac{d_i d_j}{2m}$ az i és j közötti élek számának várható értéke egy véletlen gráfban. A $q_{i,j} := a_{i,j} - \frac{d_i d_j}{2m}$ elemek alkotják G gráf M *modularitás mátrixát*. Ahhoz hogy össze lehessen hasonlítani különböző méretű hálózatok

modularitását, célszerű normalizálni a különbséget az $1/2m$ tényezővel. Így a modularitás értéke -1 és 1 közé eső szám lesz.

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, a gyakorlatban az erős közösségi szerkezetű hálózatok modularitása általában a 0,3 és 0,7 közötti tartományba esik.

2.1.2. Modularitás maximalizálási probléma

Newman [12]-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.

A modularitás maximalizálási probléma NP-nehéz, nem valószínű, hogy létezik olyan hatékony algoritmus, amely mindig megtalálja az optimális közösségszerkezetet polinomiális időben. Exponenciális idő végig menni az összes lehetőségen, hogy megtaláljuk a maximális modularitást, ami a gyakorlatban megvalósíthatatlan nagy hálózatok esetében. Ennek elkerülésére a közelítő megoldást adó módszereket haszálhatjuk, mint a szimulált hűtés és a genetikus algoritmusok.

2.1.3. Egészértékű lineáris programozási modell leírása

[11]-ben egészértékű lineáris programozási modellként fejezték ki a modularitás maximalizálási problémát, ezt a módszert implementáltam. Az egészértékű programok megoldása is NP-nehéz, azonban kis gráfokon az optimális megoldást adja vissza.

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

Legyen $I = \{(i,j) \in I \mid i < j\}$ és $q_{i,j} = a_{i,j} - \frac{d_i d_j}{2m}$ modularitás mátrixa, elemei konstansok, ezért lesz a célfüggvény lineáris.

Cél a modularitás maximalizálása, ezért a célfüggvény:

$$max\ \frac{1}{2m}\sum_{(i,j)\in I} q_{ij}(1-x_{ij}) \tag{2.3}$$

$$x_{ij} + x_{jk} - x_{ik} \geq 0 \qquad \forall i < j < k$$

$$x_{ij} - x_{jk} + x_{ik} \geq 0 \qquad \forall i < j < k$$

$$-x_{ij} + x_{jk} + x_{ik} \geq 0 \qquad \forall i < j < k$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in I$$

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.

2.2. TSP alapú

Az *utazó ügynök probléma (Travelling Salesman Problem, TSP)* az egyik legjobban kutatott probléma az optimalizálás területén, [15]–ben javasoltak a TSP-re alapuló közösségkereső algoritmust. Az utazó ügynök probléma célja, 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.

2.2.1. 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 [16] 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 p_i oldalra érkezünk a következőképpen definiálták:

$$PR^t(p_i) = (1-d)\frac{1}{N} + d\sum_{j=1}^{N} L(p_j p_i) PR^{t-1}(p_j) \tag{2.4}$$

, ahol $L(p_i, p_j)$ annak a valószínűsége, hogy p_j-ből p_i-be lépünk, és a $d \in [0, 1]$ 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:

$$PR^0(p_i) = \frac{1}{N} \tag{2.5}$$

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 C_i közösségének $v_i \in V$ 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 $PRF(v_i)$ vektornak azt kell megmutatnia, hogy melyik csúcsban állunk meg a legnagyobb valószínűséggel, ha a kezdő csúcs egy meghatározott v_i pont.

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

$$PRF^t(v_i) = \begin{pmatrix} prf_{i1}^t \\ prf_{i2}^t \\ \vdots \\ prf_{iN}^t \end{pmatrix} \tag{2.6}$$

, ahol $prf_{i1}^t, j = 1, 2, \ldots, N$ annak a valószínűsége, hogy a v_i csúcsból indulunk, és a v_j csúcsban állunk meg *t* lépés után. Ha v_i a megadott kezdő csúcs, akkor $PRF(v_i)$ kezdőértéke ($PRF^0(v_i)$) egy vektor, ahol az i-edik elem 1, minden más elem 0.
A PR alapján a *PRF* mátrixot a következőképpen lehet kiszámítani:

$$PRF^t = T \cdot PRF^{t-1} \; t = 2, 3, \ldots \tag{2.7}$$

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

Azt mondtuk, hogy a közeli csúcsok klaszteresződnek, 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 v_i csúcsból indulva hasonló valószínűséggel érkezünk v_k ($k = 1, 2, \ldots, N$) csúcsba, mint ha a v_j csúcsból indulnánk, hiszen azonos közösséghez tartoznak.

Ahogy *PR,* úgy *PRF* is egy konkrét értékhez konvergál, ezért, ha *t* elég nagy, akkor v_i és v_j csúcsok $PRF^t(v_i)$ és $PRF^t(v_j)$ 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.

2.2.2. 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:

$$d(v_i v_j) = \|PRF(v_i) - PRF(v_j)\|_2 = \sqrt{\sum_{k=1}^{N}(prf_{ik} - prf_{jk})^2} \tag{2.8}$$

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:

$$PP = \{v_j \in V | \exists v_i \in V : prf_{ij} \geq prf_{ik}, ahol\ k = 1, 2, \dots, N\} \tag{2.9}$$

Vagyis azon csúcsok halmaza, ahol a *PRF* érték a legnagyobb az adott csúcs (v_i) *PRF* vektorában.

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:

$$PRD(v_i, v_j) = \sqrt{\sum_{v_k \in PP}(prf_{ik} - prf_{jk})^2} \tag{2.10}$$

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.

2.2.3. 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 [30] egy Python könyvtár a TSP probléma megoldásához. Egzakt és heurisztikus metódusai egy távolság mátrixot várnak paraméterként. A távolság mátrix egy *n* x *n* 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 x_0 paramétert nem adjuk meg.

2.2.4. 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, [15]-ben a távolságok átlagának és szórásának összegét javasolták küszöbértéknek.

$$\delta = \mu + \sigma \tag{2.11}$$

$$\mu = \frac{1}{N}\sum_{i=1}^{N} D_i \tag{2.12}$$

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(D_i - \mu)^2}, \tag{2.13}$$

ahol D_i az optimális út i-edik és az I + 1-edik csúcsának *PRD* értéke.

3. 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.

3.1. 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*).

$$NMI(A,B) = \frac{-2\sum_{i=1}^{C_A}\sum_{i=1}^{C_B} n_{ij}\log\left(\frac{n_{ij}n}{n_{i.}n_{.j}}\right)}{\sum_{i=1}^{C_A} n_{i.}\log\left(\frac{n_{i.}}{n}\right)+\sum_{i=1}^{C_B} n_{.j}\log\left(\frac{n_{.j}}{n}\right)}, \tag{3.1}$$

ahol n a hálózat pontjainak száma, C_A az alapigazság közösségek száma, C_B a talált közösségek száma, n_{ij} azon *i* valós alapigazság közösségben lévő csúcsok száma, amelyek megtalálhatók a *j* talált közösségben is, n_i az *i* alapigazság közösségben lévő csúcsok száma, n_j 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 pontosság 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.

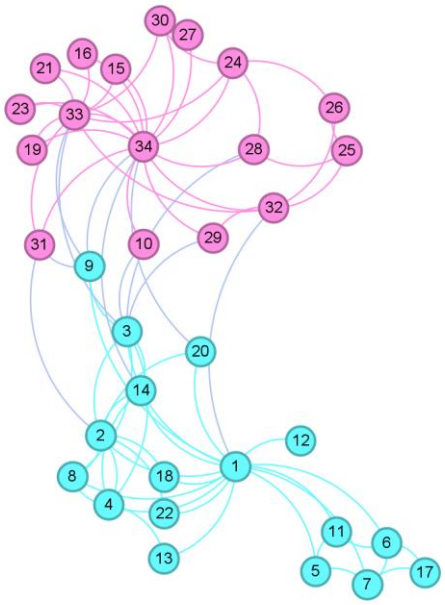
3.2. Valós világbeli gráfok

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 [18] 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

3.1. táblázat. A teszteléshez felhasznált való világbeli hálózatok. Az *n* a csúcsok száma, az *m* az élek száma.

Zachary karate klub [20] 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 meghatározott közösségszerkezetet ehhez tudjuk viszonyítani. Az 3.1. ábrán a az 1-es pont a klub oktatója, a 34-es pont a klub adminisztrátora.



3.1. ábra. Zachary karate klubja gráffal ábrázolva a klub kettéválása után. A közösségeket külön színnel jelöljük.

A delfin szociális hálózat [21,22] a Doubtful Sound palackorrú delfinjei közötti gyakori társulásokat írja le. 1995 és 2001 között minden alkalommal, amikor delfinrajjal találkoztak a fjordban, minden felnőtt egyedet lefotózták, és azonosították a hátúszójuk alapján. A hálózat csúcsai delfineket reprezentálnak. Két delfint akkor köt össze él, ha gyakrabban, láttak együtt, mint ahogy az véletlenszerűen várható lenne.

A Les Miserables hálózat [23] csúcsai Victor Hugo Nyomorultak című regényének szereplői. Ha két szereplőt összeköt él, az azt jelenti, hogy a szereplők megjelennek ugyanabban a fejezetben.

A College Football gráf [24] 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 [24] 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.

3.3. 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 [28]-ban leírt algoritmus erre a problémára nyújt megoldást.

A mesterséges hálózatainkat az algoritmusuk implemetációjával [27] 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 = 10, maxk = 50, minc = 20, maxc = 50 értékeket választottam. A mu mixing paraméter 0,1 és 0,2 értékeket kapott. μ = 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 μ = 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.

3.4. IP alapú eljárás

Az IP alapú modell-t AMPL-lel (A Mathematical Programming Language) [25] 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 x_{ij} értékeit.

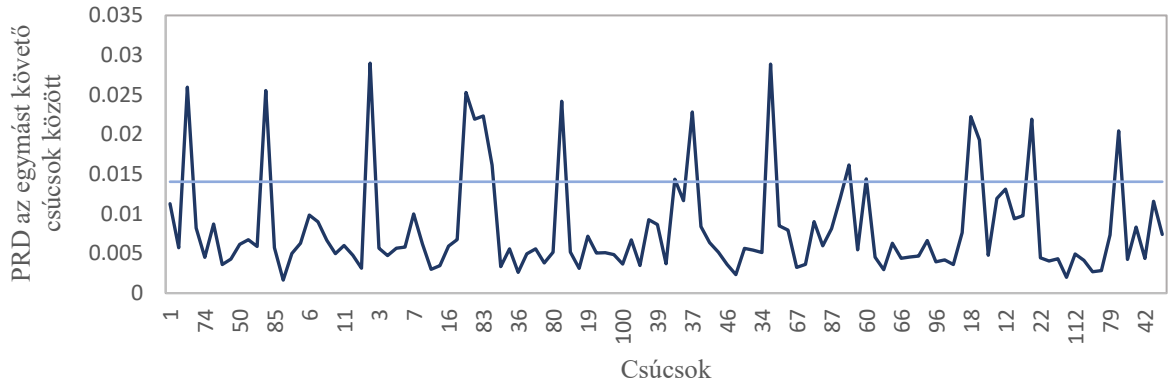
A futási idő méréséhez AMPL-ben a „display _ampl_elapsed_time +_solve_elapsed_time;” parancsot használtam a futtató scriptben. Az „_ampl_elapsed_time” az AMPL folyamat által eltöltött idő másodpercben, a megoldó idejét nem számítva, a „_solve_elapsed_time” a "solve" meghívása és a megoldás visszakapása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

hálózat	*n*	*Q*	*c*
Zachary's karate club	34	0,235	4
Dolphin social network	62	0,275	5
Les Miserables	77	0,292	10
Books about U.S. politics	105	0,270	5
College Football	115	0,307	6

3.2. táblázat. Valós világbeli hálózatok. Az *n* a csúcsok száma, a *Q* az optimális modularitás, a *c* a közösségek száma.

3.5. 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 verzióját 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.



3.2. ábra. A TSP lokális kereső eljárással kapott útban szomszédos csúcsok PRD értékei az amerikai futball hálózaton.

A 3.2. ábrán a sötétkék vonal jelöli az úton az egymást követő csúcsok PRD értékeit, a vízszintes vonal a vágás küszöbértéke, a felette lévő értékek mentén vágjuk el a közösségeket.

A futási időt Pythonban a timeit modul [31] 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.

3.6. Eredmények összehasonlítása

A 3.3. és a 3.4. táblázat foglalja össze a az algoritmusok eredményeit NMI pontosság és a talált közösségek száma (IP-*c,* SA-*c,* LS-*c*) szerint. Minél nagyobb a hálózat, a TSP-t megoldó heurisztikák annál magasabb NMI értéket értek el, ezek közül a szimulált hűtés teljesített egy kicsit jobban.

Hálózat	*n*	*c*	IP	IP-*c*	SA	SA-*c*	LS	LS-*c*
Karate	34	2	0,5878	4	0,2654	4	0,2946	4
Dolphins	62	-	-	5	-	7	-	7
Les Miserables	77	-	-	6	-	4	-	4
Books	105	3	0,5603	5	0,5952	14	0,5922	14
College Football	115	12	0,8903	10	0,9326	17	0,9244	15

3.3. táblázat. Az egészértékű programmal, a TSP szimulált hűtéssel és a TSP helyi kereső eljárással elért NMI értékek a valós hálózatokon. Az *n* a csúcsok száma, a *c* az alapigazság közösségek száma.

A mesterséges hálózatok esetében az optimális modularitás által meghatározott közösségszerkezet NMI pontossága minden esetben 1 lett vagyis megegyezik a hálózatok beépített közösségszerkezetével. A 3. hálózatban a TSP heurisztikák is megtalálták az optimális partícionálást, a többi hálózaton közelítő megoldást kaptunk.

Hálózat	*n*	γ	β	*k*	μ	*c*	IP	IP-*c*	SA	SA-*c*	LS	LS-*c*
1	100	2	1	10	0.1	2	1	2	0,7761	5	0,7704	5
2	100	2	1	10	0.15	2	1	3	0,8741	5	0,7736	8
3	100	3	2	10	0.1	3	1	3	1	3	1	3
4	100	3	2	10	0.15	3	1	3	0,9173	5	0,9204	5

3.4. táblázat. Az egészértékű programmal és a TSP szimulált hűtéssel és helyi kereső eljárással elért NMI értékek a generált hálózatokon. Az *n* a csúcsok száma, a β a fokszámsorozat mínusz exponense, a γ a közösségek méreteloszlásának mínusz exponense, a *k* az átlagos fokszám, a μ a mixing paraméter.

A futási időkről a 3.5. táblázatban és a 3.6. táblázatban számolok be. A problémát a leggyorsabban a TSP lokális kereső heurisztika oldotta meg, ami nem meglepő, hiszen a két heurisztika közül általában ennek a kisebb az időigénye. A TSP szimulált hűtés lassabb, de kisebb az esélye, hogy lokális minimumban ragad. A lineáris programnak sikerült a legtöbb idő alatt meghatároznia a közösségeket, hiszen a heurisztikákkal ellentétben addig futott, ameddig meg nem találta az optimális modularitással rendelkező közösségszerkezetet. Minél nagyobb gráffal teszteltünk, annál több időt vett igénybe.

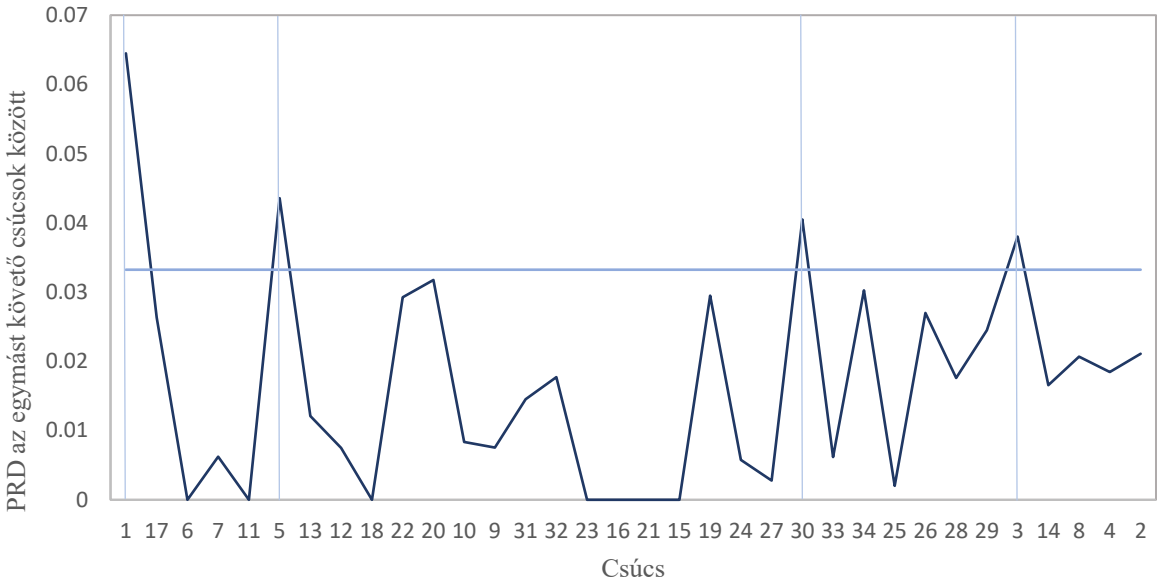
Hálózat	n	IP	TSP-SA	TSP-LS
Karate	34	0,328	0,494	0,052
Dolphins	62	28,281	1,513	0,228
Les Miserables	77	6,078	2,452	0,387
Books	105	53,625	3,871	1,171
College Football	115	80,766	5,492	1,605

3.5. táblázat. Egészértékű program (IP), TSP szimulált hűtés (TSP-SA) és TSP lokási kereső módszer (TSP-LS) futási ideje másodpercben a valós világbeli hálózatokon tesztelve. Az n a csúcsok száma.

Hálózat	n	γ	β	k	μ	IP	IP-Q	TSP-SA	TSP-LS
1	100	2	1	10	0.1	46,407	0,210	4,069	0,770
2	100	2	1	10	0.15	13,796	0,250	4,990	1,019
3	100	3	2	10	0.1	21,203	0,274	3,737	0,845
4	100	3	2	10	0.15	28,641	0,257	4,105	0,731

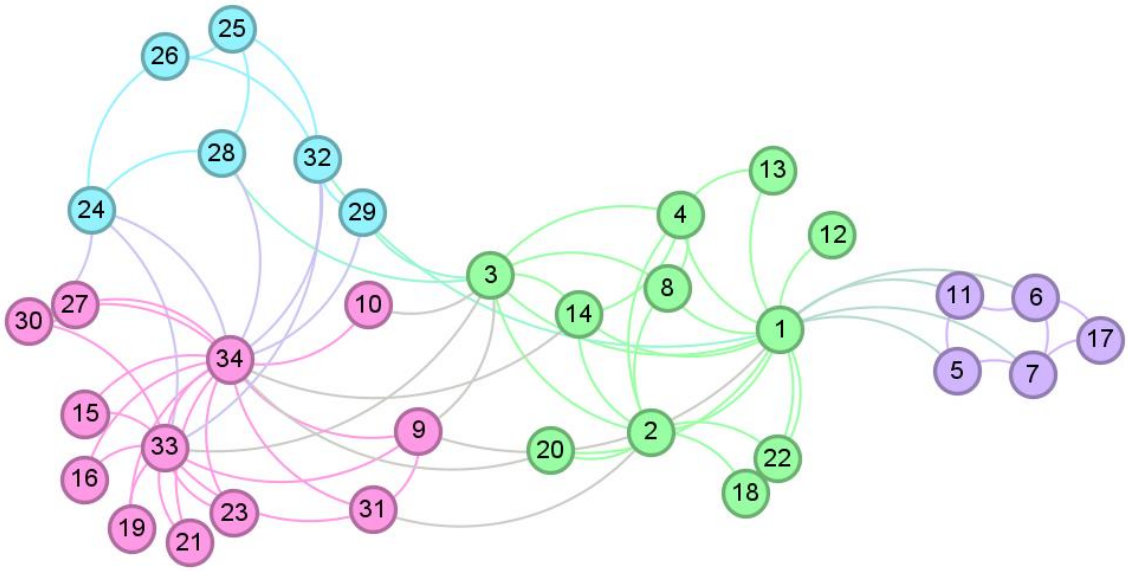
3.6. táblázat. IP, TSP-SA és TSP-LS futási ideje másodpercben a mesterséges hálózatokon tesztelve. Az n a csúcsok száma, a β a fokszámsorozat mínusz exponense, a γ a közösségek méreteloszlásának mínusz exponense, a k az átlagos fokszám, a μ a mixing paraméter, és az IP-Q az IP megoldásának modularitása.

Végül a 3.3. és 3.4. ábrákon megjelenítjük a karate klub közösségszerkezetét az IP és a TSP lokális kereső algoritmus eredményeként.



3.3. ábra. A TSP lokális kereső eljárással kapott útban szomszédos csúcsok *PRD* értékei a karate klub gráfon.

A 3.3. ábrán a sötétkék vonal jelöli az úton az egymást követő csúcsok *PRD* értékeit. A vízszintes vonal a vágás küszöbértéke, a függőleges vonalak a közösségeket határolják el. Az ábrán látjuk, hogy négy távolság van a küszöbérték felett, ezek mentén vágjuk el a kört négy közösségre.



3.4. ábra. A karate klub IP modell megoldásával kapott közösségszerkezete. A csúcsok színezésével különítjük el a közösségeket.

Összefoglalás

A dolgozatban bemutatott algoritmusokat leteszteltem való világbeli és számítógéppel generált hálózatokon. Az eredmények pont azt mutatják, amire a feldolgozott szakirodalom alapján számítottam. Az egészértékű programozási modellek kis gráfokra korlátozódnak, azonban ha meg tudják oldani a problémát elfogadható időben, az optimális megoldást adják vissza. Ezt az is igazolja, hogy a beépített közösségi szerkezettel generált hálózatokban minden esetben megtalálták a beépített közösségeket. A TSP heurisztikák az esetek nagy részében csak közelítő megoldást találtak, azonban a futási idő szerint sokkal jobban teljesítettek, mint az egészértékű program. Ez nem meglepő, hiszen a heurisztikák nem garantálják az optimális megoldást, és olyan nagy méretű hálózatokra találták ki őket, amelyeket nem lehet egzakt módszerrel elfogadható időben megoldani. A a TSP megoldók közül a helyi kereső eljárás oldotta meg gyorsabban a problémát, viszont a pontosságban nem volt jelentős eltérés a szimulált hűtéstől.

Irodalomjegyzék

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

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

[3] https://www.inf.u-szeged.hu/~london/Halozatok/halozat5.pdf (2023. december 9.)

[4] https://www.inf.u-szeged.hu/~london/Halozatok/halozat1.pdf (2023. december 9.)

[5] https://www.math.u-szeged.hu/~katai/diszmat2/eavaz/3grafelmelet.pdf (2023 .december 9.)

[6] https://www.khanacademy.org/computing/computer-science/algorithms/graph-representation/a/representing-graphs (2023. december 9.)

[7] https://www.inf.u-szeged.hu/~pluhar/oktatas/grafalg.pdf (2023. december 9.)

[8] https://www.inf.szte.hu/~rfarkas/Alga17/9_ElemiGrafalgoritmusok.ppt (2023. december 9.)

[9] Barabási Albert-László, A hálózatok tudománya, Libri, 2016

[10] A. Ferdowsi and A. Khanteymoori, Discovering communities in networks: A linear programming approach using max-min modularity. In: 2021 16th Conference on Computer Science and Intelligence Systems (FedCSIS), pp. 329–335. IEEE (2021)

[11] G. Agarwal and D. Kempe, Modularity-maximizing graph communities via mathematical programming, The European Physical Journal B, vol. 66, no. 3, pp. 409–418 (2008)

[12] M. Newman, Fast algorithm for detecting community structure in networks, Physical Review E, 69, 066133 (2004)

[13] M. E. Newman, Modularity and community structure in networks, Proceedings of the national academy of sciences, vol. 103, no. 23, pp. 8577–8582 (2006)

[14] M. Newman, M. Girvan, Finding and evaluating community structure in networks , Phys. Rev. E 69, 026113 (2004)

[15] Jiang, Z., Liu, J., Wang, S. Traveling salesman problems with PageRank Distance on complex networks reveal community structure. Physica A: Statistical Mechanics and its Applications, 463, 293-302 (2016)

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

[17] 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)

[18] https://www-personal.umich.edu/~mejn/netdata/ (2024. május 18.)

[19] M. Girvan and M. E. Newman, Community structure in social and biological networks, Proceedings of the national academy of sciences, vol. 99, no. 12, pp. 7821–7826 (2002)

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

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

[22] D. Lusseau, The emergent properties of a dolphin social network, Proc. R. Soc. London B (suppl.) 270, S186-S188 (2003)

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

[24] M. Newman, Proc. Natl. Acad. Sci. USA 103, 8577 (2006)

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

[26] https://www.santofortunato.net/resources (2024. május 17.)

[27] A. Lancichinetti, S. Fortunato, Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities, Physical Review E 80, 016118 (2009)

[28] A. Lancichinetti, S. Fortunato and F. Radicchi, Benchmark graphs for testing community detection algorithms, Physical Review E 78, 046110 (2008)

[29] https://networkx.org/ (2024. május 18.)

[30] https://github.com/fillipe-gsm/python-tsp (2024. május 18.)

[31] https://docs.python.org/3/library/timeit.html (2024. május 18.)

Nyilatkozat

Alulírott Juhász Dóra gazdaságinformatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Számítógépes Optimalizálás Tanszékén készítettem, gazdaságinformatikus BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Diplomamunka Repozitóriumában tárolja.

2024. május. 18.

Juhász Dóra

Köszönetnyilvánítás

Szeretném megköszönni a témavezetőmnek, Dr. Vinkó Tamásnak, hogy egy számomra érdekes témát ajánlott, amellyel egyetemi tanulmányaim során eddig nem volt lehetőségem mélyebben foglalkozni. Hálás vagyok a heti konzultációkért, a türelméért és a sok tanácsért, amit a szakdolgozatom elkészítéséhez kaptam.