Forgalom igény tudatos hálózat tervezés minimális torlódással és úthosszal

Tudáskezelő rendszerek IV. labor összefoglaló

Szecsődi Imre

2019

Tartalomjegyzék

1. Bevezetés **2**
1.1. Labor célja 2
1.2. Laborban megvalósított munka 2

2. Modell **3**
2.1. Az új algoritmusok 3
2.1.1. Egobalance 3
2.1.2. Huffman fa 4
2.1.3. Sorfolytonos fa 4

3. Megvalósítás **5**
3.1. Adatszerkezetek 5
3.2. Modell 5

4. Teszt eredmények **7**
4.1. Tesztelés menete 7
4.2. Eredmények 7
4.2.1. Átlag Súlyozott úthossz 7
4.2.2. Átlag torlódás 9

5. Összefoglalás **12**
5.1. Labor eredménye 12

6. Irodalomjegyzék **13**

1. fejezet

Bevezetés

1.1. Labor célja

A labor célja a korábban már bemutatott cikkben[4] szereplő hálózat megépítése és annak kiegészítése. Az elkészült keresztrendszer kiegészítése még egy fa építési stratégiával, ez pedig a Véletlenszerűen felépített fa. Továbbá még különböző metrikák bevezetése, ami segítségével pontosabb képet kapunk arról, hogy különböző helyzetekben milyen eredményeket eredményeznek az algoritmusok.

1.2. Laborban megvalósított munka

A labor ideje alatt a már elkészült keretrendszer adta az alapot, ami segítségével tesztelhető a szerzők által felvázolt és három saját fejlesztésű algoritmus. A keretrendszer Python [3] nyelven íródott. A véletlen gráfok generálására a NetworkX külső csomag volt használva[2]. Az adatok elemzése RapidMiner-ben történt[1].

2. fejezet

Modell

2.1. Az új algoritmusok

2.1.1. Egobalance

Az első algoritmus majdnem teljes mértékben megegyezik az eredetivel. A cikk szerzői az Egofa algoritmus vázlatos összefoglalásában használtak egy csere lépést. A csere lépés lényege, hogy mikor megépítettük a fát egy magas csúcsra, akkor a fában minden szomszédja megjelenik. A szomszédok között megtalálhatók a magas-magas fokszámú kapcsolatok, amiket ki kell cserélni a segítő csúcsokra. Három esetet különböztetünk meg itt, attól függően hol helyezkedik el a segítő csúcs a fában. Első eset, mikor a segítő csúcs nem szerepel a fában, ilyenkor a magas csúcsot ki kell cserélni a segítőre. Ebben az esetben nem kell semmi kiegészítő lépést csinálni, mivel a gyerekeket átveszi a segítő. A következő két eset, mikor a segítő is része a fának. Attól függően mennyire közel vannak az érintett csúcsok a fa gyökeréhez más-más gyerek csúcsokat kell újra elhelyezni. Mikor a segítő csúcs közelebb van a gyökérhez, akkor töröljük a magas pontot a fából, ha voltak gyerekei a törölt pontnak, akkor azoknak új szülő csúcsot kell találni. Ellenkező esetben, mikor magas pont helyezkedik el közelebb a forráshoz, akkor a segítő csúcs átveszi a helyét és gyerekeit, majd a segítő leveleinek kell új szülő csúcsot találni. Ezt úgy oldja meg az eredeti algoritmus, hogy a nehezebb levél csúcs lesz az új szülő és a könnyebb csomópontnak ő lesz a szülője.

Egy lehetséges eset ilyenkor, hogy az új szülő csúcsnak már ki van töltve mindkét levele, ekkor a szülő könnyebb levele fog egyel lejjebb szintre kerülni. Ezt a folyamatot addig ismételjük, még minden levél nem kerül egy megfelelő helyre. Az Egobalance logaritmus a függő pontok újra elhelyezését az Egofára bízza, azaz újra elosztásra kerülnek. Ezzel elméletben mindig arra törekszik az algoritmus, hogy optimális legyen torlódásra nézve az új fa. Lényegi különbség úthossznál jelentkezik az eredetivel szemben.

2.1.2. Huffman fa

A eddigi két algoritmus az Egofát[4] használta, ami optimális torlódásra nézve, de mivel a hálózat nem csak torlódás szempontjából van vizsgálva, ezért meg kell vizsgálni a másik aspektust is, az optimális úthosszt. A Huffman kódolásnál[5] használt fa tulajdonsága, hogy átlagosan rövidek legyenek az utak attól függően milyen gyakori egy elem. A cl-Dan probléma is ezt a tényezőt használja, ezért kitűnően lehet használni ezt a fát a hálózat alapjának. Egyetlen probléma a Huffman fával az, hogy a belső csomópontok nem tartalmaznak számunkra hasznos információt. Ezért szükséges egy kiegészítő lépés, ami segítségével belső pontok is ki lesznek töltve, azaz esetünkben csomópontokat fognak reprezentálni mint a levelek.

Az algoritmus első része teljesen megegyezik a Huffman kódolással. Rendezzük sorba növekedően a pontokat, és kettesével vonjuk össze őket, még nem kapunk egy teljes fát. Mint az Egofáknál, úgy a Huffman fánál is a legfelső szinten n darab csúcsot tudunk a forrás pontra kapcsolni. Az összes többi alacsonyabb pont pedig marad bináris.

A belső csúcsok kitöltésére a gyökér ágain a következő algoritmust végezzük el:

1. Gyűjtsük össze a levelet az ágon, a levelek tartalmazzák a számunkra hasznos pontokat

2. Az ág gyökerétől indulva sorfolytonosan helyezzük el a leveleket, ahol a valószínűségek csökkenő sorrendben vannak rendezve

Felmerülhet a kérdés, hogy miért nem a legnehezebb levél jön fel mindig? Ez azért van, mert a Huffman kódolásnál megtörténik az eset, hogy két csomópont összesített értéke megegyezik egy harmadikkal. Ez egy olyan fát eredményez, ahol az egyik oldalon egy nehéz csúcs, a másik oldalon pedig két könnyű csúcs szerepel. A naiv megoldás azt eredményezi, hogy az a nehéz pont lesz a belső csúcs és az ág ahonnan jött megüresedik. A könnyebb fa levelei nem fognak ágat változtatni, annak ellenére, hogy megüresedett az ág feljebb, ezért hosszú egyenes utak jöhetnek létre. Ennek kiküszöbölésére van a sorfolytonos algoritmus, ahol garantálni lehet, hogy fa egyik belső pontja sem marad kitöltetlen.

2.1.3. Sorfolytonos fa

Mint láthattuk korábban a Huffman fánál, a levelek felfelé mozgatását sorfolytonosan valósítottuk meg. A sorfolyonos fa is hasonló elvet követ. Lényegi különbség a kettő fa között, hogy a Huffman kódolási algoritmust kihagyjuk, és egyenest sorfolytonosan rakjuk fel csúcsokat a fa építésekor. Ezzel mindig a legkisebb fákat kapjuk, de ez a torlódást egyáltalán nem veszi figyelembe.

3. fejezet

Megvalósítás

3.1. Adatszerkezetek

A modell alapját pár egyszerű alaptípus adja. Ezek rendre a következők:

- **Vertex** - az általános gráf csúcs
- **Node** - az *Egófák* készítésekor használt csúcs, ami tartalmazza a valószínűségét annak, hogy a forrás csomópont mekkora valószínűséggel fog kommunikálni a másik *Node* csúccsal
- **HuffmanDanNode** - a Huffman fa csomópontjait reprezentáló osztály, minden tulajdonsága megvan mint a Node-nak, csak még kiegészül egy útvonallal, amit a Huffman fa építése után lesz meghatározva
- **Edge** - a gráf pontjait összekötő él reprezentációja, ami *Vertexet* vár paraméterként, és tartalmazza a kommunikációs valószínűséget, hasonlóan mint a *Node*
- **Tree** - ami adja az alapját majd a útvonal tervezési sémának. A fának két fajtája van megvalósítva:
  - **EgoTree** - a Δ fokú Egofa, ahol a gyökérnek legfeljebb Δ levele lehet
  - **HuffmanDanTree** - a Δ fokú Huffman fa, ahol a gyökérnek legfeljebb Δ levele lehet és a belső csúcsok állhatnak üresen

3.2. Modell

A **Network** osztály megváltozott az előző megvalósításhoz képest, absztrakt osztályra lett átalakítva és a belőle leszármazó osztályok valósítják meg a különböző

algoritmusokat. A leszármaztatott osztályoknak egyetlen függvényt kell csak megvalósítani, ami a fa építései stratégiát tartalmazza. Ennek segítségével nagyon egyszerűen lehet új algoritmust illeszteni a már meglévő rendszerhez.

A bemeneti konfiguráció kiegészült egy új gráf típussal, ami a csillag gráf. Csillag gráfnak nevezzük azt a gráfot, ahol az összes pont egy ponthoz kapcsolódik. A konfiguráció további egységesítésen esett át, minden véletlen gráf szerkezete megegyezik, ami a következő:

- `graph` - Gráf típusa, lehetséges értékek:
  - `barabasei-albert` - Barabási-Albert gráf
  - `erdos-renyi` - Erdős-Rényi gráf
  - `star` - Csillag gráf
- `vertex_num` - Csomópontok száma
- `dan` - A maximális fokszám a magas fokú csúcsokra
- `constant` - Konstans érték ami a gráf paraméterét adja, ezek rendre a következők:
  - Barabási-Albert gráf esetén azt határozza meg, hogy az új csúcs mennyi már a gráfban szereplő csúcshoz kapcsolódjon
  - Erdős-Rényi gráf esetén az él valószínűségének meghatározásra használjuk a következő képlet segítségével

$$p = \frac{constant}{vertex\_num}$$

  - Csillag gráf esetén pedig a csillagok számát adjuk meg

4. fejezet

Teszt eredmények

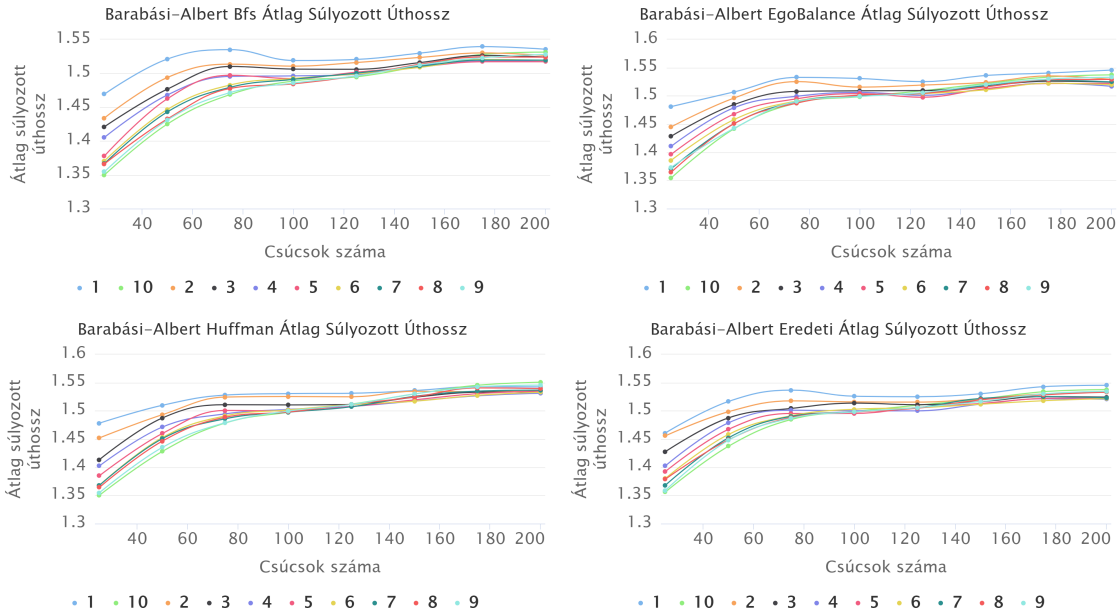
4.1. Tesztelés menete

A tesztelés során mindegyik véletlen gráf volt használva, és minden algoritmus ugyanazon a bemeneti mátrixon dolgozott a egyértelmű összehasonlítás érdekében. A mérések a konstans alapján lettek kiértékelve, mivel az a fő érték, ami alapján változik a gráf ritkasága. A skálázhatóság tesztelésére különböző számú csúcs volt használva, ezek rendre: 11, 15, 35, 50, 75, 100, 125, 150, 175 és 200. A skálázhatósághoz még hozzátartozik a maximum fokszám, hogy mennyi gráf csúcs tud egyszerre kommunikálni, ezért az is meg van adva. A valóságban van egy fizikai határ, hogy mennyi kliens tud csatlakozni egyszerre, ezért az értékek így lettek megválasztva, 10, 16, 24 és 48. A konkrét számú pont mellett meg volt vizsgálva az is, hogy valóságban mekkora szorzóra van szükségünk. A cikk ad egy felső becslést ami $\Delta = 12\rho$, ezért a teszt során figyelve volt az ajánlott és szükséges fokszám. Az ajánlott fokszám értékei a következők voltak $\Delta \in [1\rho, 2\rho, 4\rho, 6\rho, 8\rho, 10\rho, 12\rho]$. Az így megadott paraméterek alapján 52800 teszt lett lefuttatva.

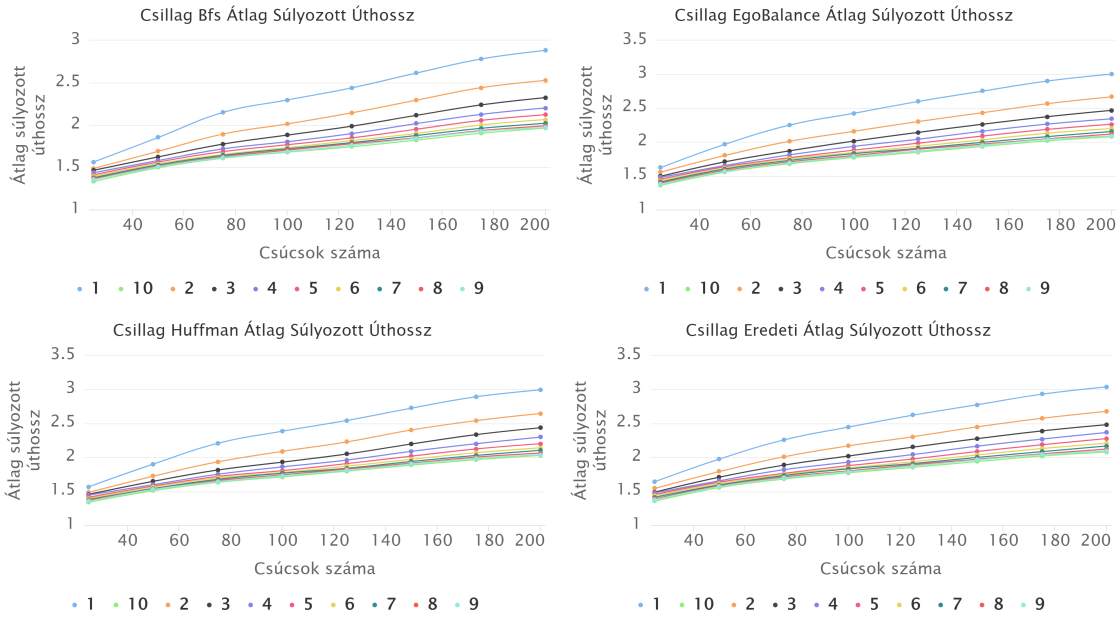
4.2. Eredmények

4.2.1. Átlag Súlyozott úthossz

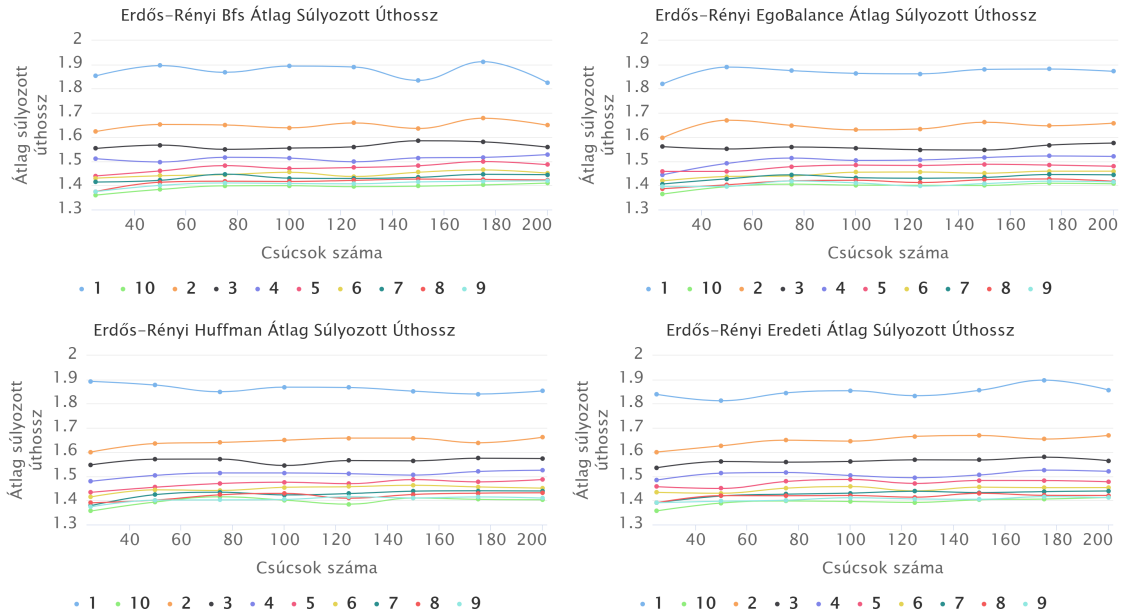
A következő grafikonok ábrázolják az eredményeket úthosszra nézve. Megfigyelhető, hogy az összes véletlen gráfon hasonlóan teljesítenek az algoritmusok. A mátrix kitöltöttségének függvényével megegyezően növekedett az úthossz értéke is. Egy érdekes megfigyelés még továbbá a tény, hogy mikor kicsi volt a konstans, feltehetően ritka mátrixot eredményezve magasabb eredményeket kapunk. Ahogy növeljük a konstanst és több érték kerül be a mátrixba, több lehetőségünk lesz olyan segítő csúcs kiválasztására amit még nem használtunk korábban, ezzel rövidítve az utakat.



4.1. ábra. Átlag súlyozott úthossz Barabási-Albert Gráf



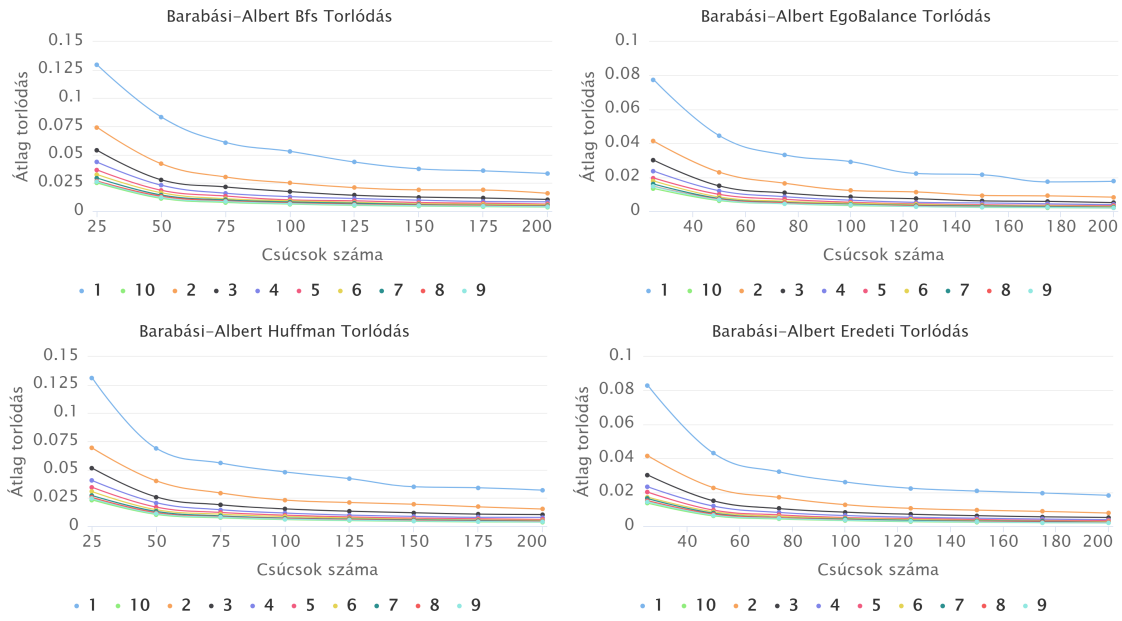
4.2. ábra. Átlag súlyozott úthossz Csillag gráf



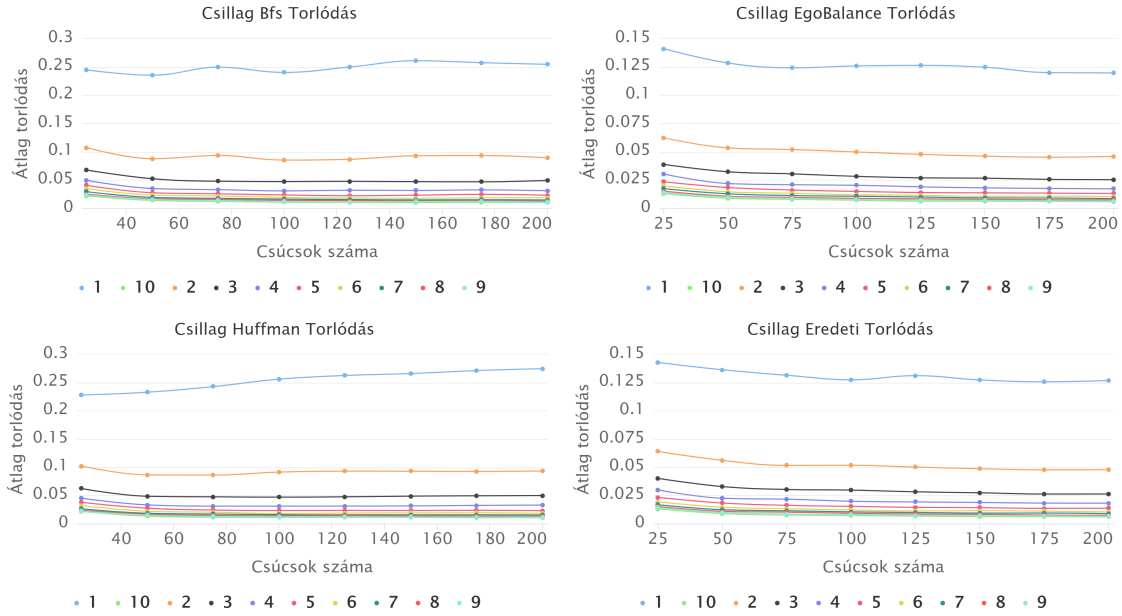
4.3. ábra. Átlag súlyozott úthossz Erdős-Rényi Gráf

4.2.2. Átlag torlódás

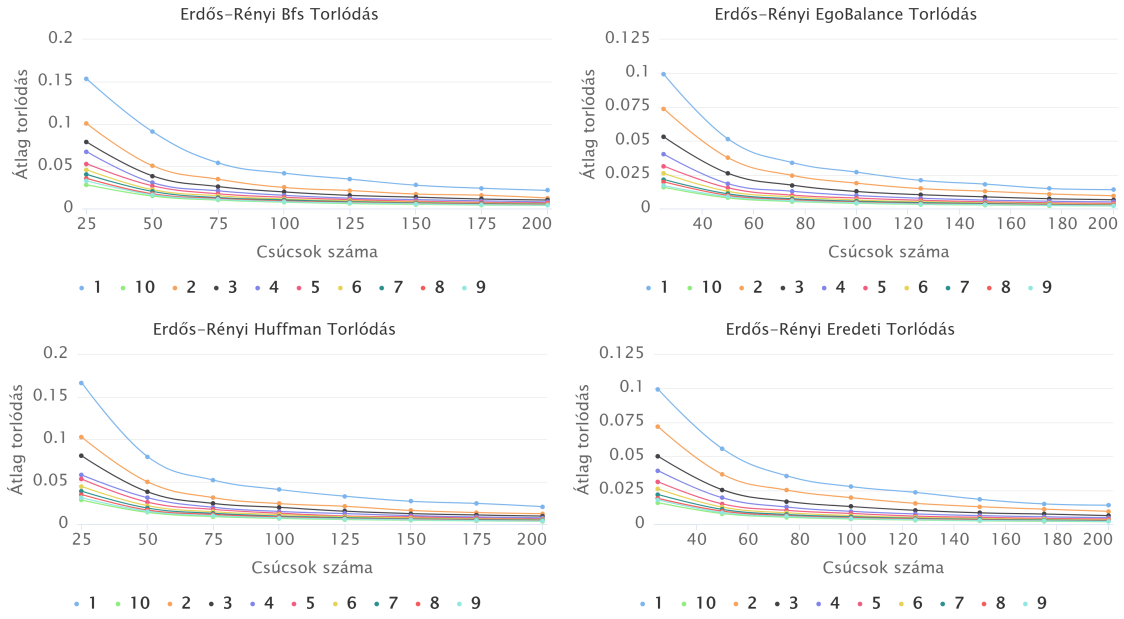
Az átlagos úthossz eredményei már kicsit érdekesebb eredményt mutatnak. Első megállapítás, hogy itt egyértelműen két csoportba lehet sorolni az algoritmusokat, attól függően milyen fát használták alapnak. Az eredeti és az EgoBalance algoritmusok figyelnek a torlódás tényezőre is, ezért egyértelműen látszik a grafikonon, hogy jobban teljesítenek. A Huffman és sorfolyonos algoritmus nem veszi számításba ezt a tényezőt, ezért nem is teljesítenek olyan jól.



4.4. ábra. Átlag torlódás Barabási-Albert Gráf



4.5. ábra. Átlag torlódás Csillag Gráf



4.6. ábra. Átlag torlódás Erdős-Rényi Gráf

5. fejezet

Összefoglalás

5.1. Labor eredménye

A tesztek kiértékelésének eredménye azt mutatja, hogy mind a négy algoritmus helytálló, mert hasonló eredményeket kapunk. Az úthosszra nézve az algoritmusok teljesítménye szintén nagyon hasonló és benne van mérési hiba tartományában. A torlódásra szempontjából van jelentős különbség, a Huffman fa alapú algoritmusok nem veszik figyelembe egyáltalán a torlódást, ezért határozottan rosszabbul teljesítenek.

Két algoritmus ami hasonló módon építi fel a fát az a sorfolytonos (Bfs) és az EgoBalance algoritmus. A kettő közötti lényeges különbség, amíg az Egofa legnehezebbtől legkönnyebbig próbálja a legjobban elosztani a súlyokat, addig a sorfolytonos folyamatosan rakja fel őket. Egy ilyen kis eltérés határozottan jobb eredményhez vezet az Egofák esetén.

Az eredmények összesítésből látszik, hogy a szerzők által felvázolt Egofa egy megfelelő adatszerkezet arra a célra, hogy hálózatot építünk vele, mivel mindkét szempontól jó teljesít.

6. fejezet

Irodalomjegyzék

[1] Lightning Fast Data Science Platform for Teams | RapidMiner©.

[2] NetworkX - http://networkx.github.io/.

[3] Python - Python.org.

[4] C. Avin, K. Mondal, and S. Schmid. Demand-aware network design with minimal congestion and route lengths. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 1351–1359, April 2019.

[5] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40(9):1098–1101, September 1952.