Eötvös Loránd Tudományegyetem

Informatikai Kar

Információs Rendszerek Tanszék

Forgalom igény tudatos hálózat tervezés

Témavezető:
Lukovszki Tamás
Egyetemi docens, PhD
Információs Rendszerek tanszék

Szerző:
Szecsődi Imre
Programtervező informatikus
MSC

Budapest, 2019

Tartalomjegyzék

1. fejezet

Bevezetés

A diplomamunka a Demand-Aware Network Design with Minimal Congestion and Route Lengths [9] cikk alapján készült.

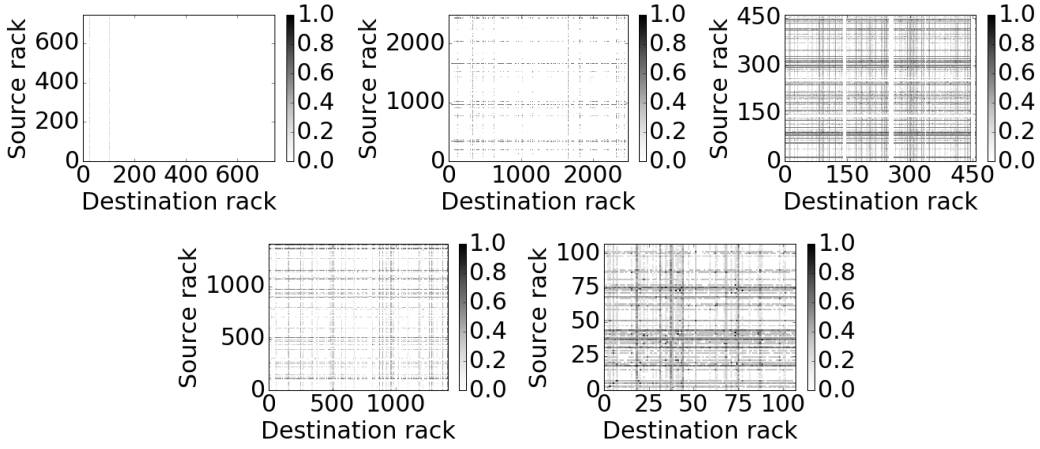
1.1. Motiváció

Az adatközpontok kritikus infrastruktúrák lettek a mai digitális társadalomban. A megnövekedett adatmennyiség miatt, hatékony hálózati megoldások szükségesek arra, hogy az adatközpontok minél hamarabb fel tudják dolgozni az adatokat. A jelenlegi hálózatok egy adatközpontban a legrosszabb esetre vannak optimalizálva, azaz az infrastruktúra által biztosított kapcsolatokon közel maximális sebességgel tudjon kommunikálni bármelyik kettő szerver. A gyakorlati tapasztalatok azt mutatják, hogy a kommunikáció túlnyomó része néhány szerverek között történik. Mikor a kommunikációs mintát ábrázoljuk egy gráfban, ahol az élek az egymással kommunikáció szervereket ábrázolják, akkor ez egy ritka gráfot eredményez.

A kommunikációs technológiák fejlődésével lehetőségünk van futás időben a fizikai hálózatok újra konfigurálására. Az ilyen technológiák segítségével lehet növelni az adatközpontok hatékonyságát. A változó adatforgalomhoz lehet alkalmazkodni és a topológiát átalakítani az szerint, hogy mekkora terhelés éri a hálózat egyes részeit. A hálózat fizikai újra konfigurálására egy megoldás a Microsoft Research ProjecToR[10], ami lézer technológia segítségével köti össze a szervereket egy adattárházban. Az új technológiák új problémákat eredményeztek és új kihívások elé állították a hálózat tervezést. Napjainkban intenzív kutatás folyik ezen a területen.

1.1.1. Microsoft adattárház adatok

Microsoft Research ProjecToR keretén belül, a szerzők adatokat gyűjtöttek Microsoft adattárházában. Adatokat kétszázötvenezer szerverről rögzítettek, ami öt prductionban használt klaszterben voltak elosztva.



1.1. ábra. Microsoft adattárház adatok klaszterenként

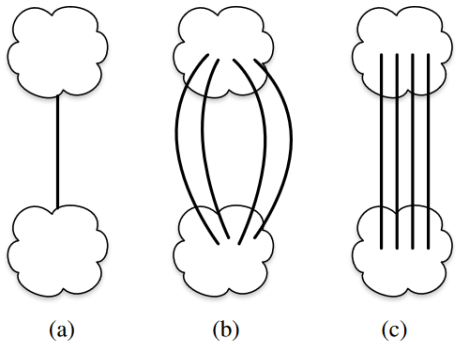
Amint a 1.1 ábrán látható, a kommunikáció főleg megadott szerverek között történik. Korábban már említve volt, az adattárházak jelenleg a legrosszabb esetre vannak tervezve, hogy bármelyik két szerver tudjon kommunikálni. Az ábrán látszik, hogy melyik szerverek között nem szükséges feltétlen direkt kapcsolatot kiépíteni, hanem elég egy már meglévő közvetett útvonalat használni, ahol kicsi a torlódás. Az ilyen hálózat megtervezéséhez először szükségünk van arra, hogy tudjuk milyen tervezési stratégiák vannak és, hogy az adattárházak milyen topológiával rendelkeznek jelenleg.

1.1.2. Hálózat tervezési stratégiák

A technika fejlődésével elérhetővé váltak eszközök arra, hogy egy adott hálózatot újrakonfiguráljunk, attól függően milyen terhelés éri. Egy hálózat korábbi kommunikációs mintái tudnak adni egy jó közelítést arra, hogyan történik a kommunikáció a szerverek között. Az előző példában láthattuk, hogy a Microsoft adattárházban milyen a forgalomeloszlást rögzítettek, és ez alapján lehet majd újratervezni a hálózatot, figyelve a terheléseloszlásra. Lehetőség szerint, periodikus újrakonfigurálással akár még nagyobb hatékonyságot is elérhetünk az adattárházban.

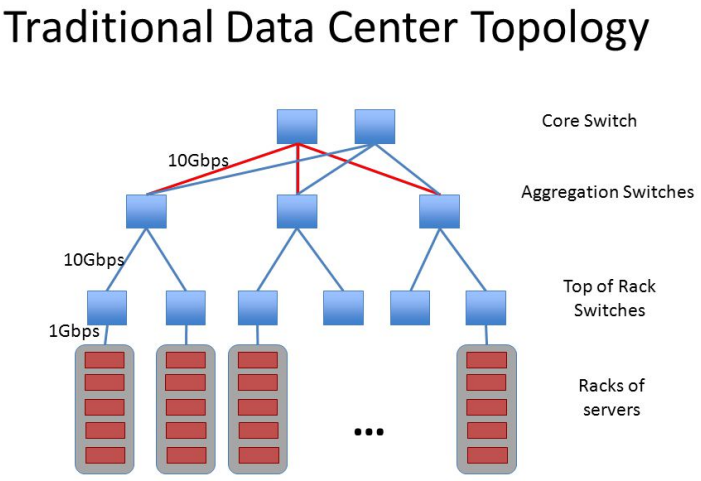
Két fő optimalizációs megközelítést fogunk megnézni a munka során, ezek a rövid úthossz és a minimális torlódás. Rövid úthossz alatt azt értjük, hogy minél kevesebb pontok kelljen áthaladnia az adatnak mielőtt az eljut a céljába, 1.2 ábrán az (a) eset. A minimális torlódás alatt pedig azt értjük, ha több adatfolyam halad át egy élen, akkor az élek egyenletesen legyenek kihasználva, ezzel csökkentve a

várakozási időt, 1.2 ábra (b) pontja. A cikk célja, egy olyan módszer bemutatása, ahol mindkét esetet figyelembe veszik és az alapján adnak egy olyan algoritmust, ami közel optimális mint úthosszra, mint torlódásra nézve, ez a 1.2 ábra (c) esete.



1.2. ábra. Hálózat tervezési stratégiák, (a) rövid utakra való optimalizálás, (b) minimális torlódásra való optimalizálás, (c) mindkét esetre való optimalizálás

1.1.3. Adattárházak hálózati felépítése



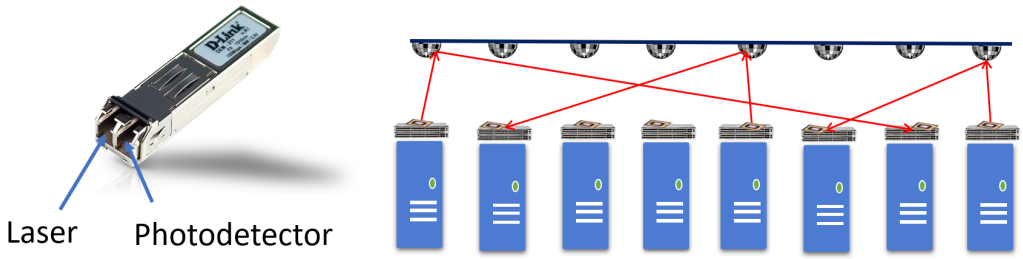
1.3. ábra. Adattárház hálózati topológiája

Mielőtt még bármi nemű átalakítást végeznénk az adattárház hálózatán, először még meg kell nézni azt, hogy hogyan is épül fel az. Az 1.3 ábrán látható a felépítés, ahol három szintre tudjuk felosztani a hálózat architektúráját[2][1]. Legfelső szinten van a Core réteg, ami adja a gerincét a hálózatnak, ezen a ponton érkeznek és távoznak a csomagok az adattárházból. A core réteg további szerepe még, hogy adatot szolgáltasson az alatta lévő Aggregációs rétegnek. Ez a réteg nagyobb csoportokra osztja az adattárházat és ezzel minimalizálják a routing táblák számát. Az

aggregációs réteg továbbá szolgáltatásokat is tud nyújtani, az alatta lévő rétegnek, mint például fűzfal, load balancing és biztonságos csatlakozás. A harmadik réteg pedig a Access réteg, ami Top of Rack switchek formájában nyilvánul meg. Ezen a ponton kapcsolódnak a szerverek fizikailag a hálózathoz. A Core rétegtől a ToR switchekig általában optikai kábellel van kivitelezve a hálózat. ToR switchtől már hagyományos módszerekkel csatlakoznak a szerverek. Egy rackben nagyon sok szerver van elhelyezve, ezért azok általában egy köztes, úgy nevezett In-Rack switchre kapcsolódnak, amik végül csatlakoznak a hozzájuk legközelebbi ToR switchhez.

1.1.4. Újrakonfigurálás megvalósítása

Az átlag hálózatok statikusan vannak konfigurálva, ezért nem sok lehetőség van arra annak megváltoztatására, ide tartoznak a hagyományos Ethernet switchek. Egy kevésbé statikus megoldás az optikai swtich, ami képes megvalósítani az újrakonfigurálást, és ezt relatív gyorsan is csinálja. Az optikai switchektől egy még gyorsabb megoldás a Microsoft Research ProjecToR. A 1.4 ábrán látható eszköz szem számára láthatatlan lézer nyalábbal küld és fogad adatokat. Az adattárház belső kommunikációjának a gyorsítására szolgál, ezért a ToR switchek vannak lecserélve, ilyen ProjecToR eszközökre. A rack tetejéről az eszköz a lézert mikrotükrökre irányítja, aminek a pontos beállíthatósága révén pontosan a megfelelő irányba tudja tükrözi tovább a nyalábot. Ennek az folyamatnak köszönhetően jelentősen gyorsabb átkonfigurálási időt érhetünk el mint az optikai switch. Váltásidő itt 12μs ami kétszázötvenezerszer gyorsabb mint az optikai switch.



1.4. ábra. ProjecToR

1.2. Diplomamunka célja

A diplomamunka célja a cikkben[9] bemutatott algoritmus implementálása, és annak alkalmazása különböző véletlen gráfokra. A kapott eredményeket összehasonlítása a megadott elméleti korlátokkal.

1.3. Laborban megvalósított munka

A labor ideje alatt elkészült egy keretrendszer, ami segítségével tesztelhető a szerzők által felvázolt algoritmus. A keretrendszer Python [6] nyelven íródott. Egy véletlen gráfok generálására a NetworkX külső csomag volt használva[5]. Az adatok elemzése pedig RapidMiner-ben történt[4].

2. fejezet

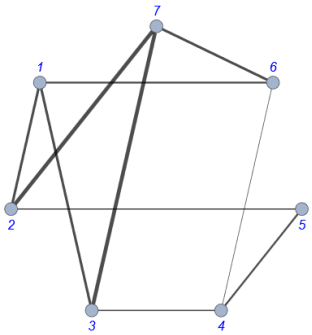
Modell

2.1. Forgalom igény tudatos hálózat tervezés probléma

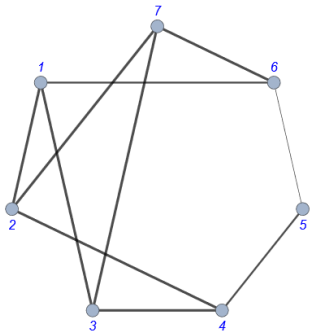
- Vegyünk egy hálózatot meghatározott számú csomóponttal
- A hálózathoz tartozik egy demand mátrix, ami leírja a valószínűségét annak, hogy i forrásból mekkora eséllyel lesz adat küldve j célba
- A cél, hogy ezen adatból egy olyan hálózati séma készítése, ami kis torlódást és rövid utakat eredményez, ez mellett még skálázható is

	1	2	3	4	5	6	7
1	0	$\frac{3}{60}$	$\frac{4}{60}$	$\frac{1}{60}$	$\frac{1}{60}$	$\frac{1}{60}$	$\frac{1}{60}$
2	$\frac{3}{60}$	0	$\frac{2}{60}$	0	$\frac{1}{60}$	0	$\frac{4}{60}$
3	$\frac{4}{60}$	$\frac{2}{60}$	0	$\frac{2}{60}$	0	0	$\frac{4}{60}$
4	$\frac{1}{60}$	0	$\frac{2}{60}$	0	$\frac{3}{60}$	0	0
5	$\frac{1}{60}$	$\frac{1}{60}$	0	$\frac{3}{60}$	0	0	0
6	$\frac{1}{60}$	0	0	0	0	0	$\frac{3}{60}$
7	$\frac{1}{60}$	$\frac{4}{60}$	$\frac{4}{60}$	0	0	$\frac{3}{60}$	0

(a) Demand distribution



(b) Optimal route length



(c) Optimal route congestion

2.1. ábra. Forgalom igény tudatos hálózat tervezés probléma, (a) demand mátrix, (b) hálózat optimális úthosszra és (c) hálózat optimális torlódásra

2.2. Formális felírás

- Adott N darab csúcspont $V = \{1, ..., N\}$, és egy kommunikációs séma M_D ami egy $N \times N$ mátrix

- A mátrix (i, j) eleméhez tartozik egy $p(i, j)$ valószínűség, ahol i a forrás csomópont és j a cél
- A bemeneti mátrix ábrázolható egy irányított G_D gráfban, ahol az élsúlyok a két pont közötti kommunikációs valószínűség
- Az algoritmus feltétele, hogy a mátrix ritka legyen
- Egy N hálózatra a torlódást és az úthosszt útválasztási sémával fogjuk definiálni
- Egy útválasztási séma az N hálózatra $\Gamma(N)$, ami Γ_{uv} utak halmaza, ahol (u, v) párok különböző utakat jelölnek
- Γ_{uv} egy útsorozat, ami összeköti az u pontot v ponttal

2.2.1. Torlódás

1. Definició. *A torlódást egy $\Gamma(N)$ útválasztási sémán a D demand mátrix segítségével írjuk fel:*

$$C(D, \Gamma(N)) = \max_{e \in \Gamma(N)} \sum_{e \in \Gamma(uv)} p(u, v)$$

2.2.2. Úthossz

2. Definició. *Az átlag súlyozott úthosszt egy $\Gamma(N)$ útválasztási sémán a D demand mátrix segítségével írjuk fel:*

$$L(D, \Gamma(N)) = \sum_{(u,v) \in D} p(u, v) \cdot d_{\Gamma(N)}(u, v)$$

ahol a $d_{\Gamma(N)}(u, v)$ az útvonal hosszát jelöli

2.2.3. Skálázhatóság

- A hálózatot skálázhatóra kell tervezni, ezért meghatározunk egy Δ konstans fokszámot, ami a maximális csatlakozások számát fogja meghatározni egy adott csomóponthoz
- N_Δ jelölje az összes Δ fokszámú gráfot, és elváruk, hogy $N \in N_\Delta$

2.2.4. Optimális torlódás

Az optimális torlódást egy hálózatra, úgy határozzuk meg, hogy a csak a torlódást vesszük figyelembe számításkor

$$C^*(D, \Delta) = \min_{N \in N_\Delta, \Gamma(N)} C(D, \Gamma(N))$$

2.2.5. Optimális úthossz

Az optimális úthosszt egy hálózatra, úgy határozzuk meg, hogy a csak az úthosszt vesszük figyelembe számításkor

$$L^*(D, \Delta) = \min_{N \in N_\Delta, \Gamma(N)} L(D, \Gamma(N))$$

2.3. cl-DAN hálózat tervezése

3. Definició. *Adott egy D demand mátrix, és egy Δ maximális fokszám, az (α, β)-cl-DAN hálózat tervezési probléma:*

- *Hogy tervezzünk egy olyan $N \in N_\Delta$ hálózatot, és egy hozzá tartozó $\Gamma(N)$ útválasztási sémát, ami közel optimális torlódásra és úthosszra is*

Az algoritmus egy felső korlátot tud adni arra, hogy mennyivel fog eltérni a megoldás az optimálistól.

- *Torlódásra:* $C(D, \Gamma(N)) \leq \alpha \cdot C^*(D, \Delta) + \alpha'$
- *Úthosszra:* $L(D, \Gamma(N)) \leq \beta \cdot L^*(D, \Delta) + \beta'$

Az alfa vessző és béta vesszők olyan tényezők aki amik függetlenek a problémától

2.4. EgoTree

- Az Egofa egy torlódásra és úthosszra optimalizált fa hálózat egy csomópontra nézve
- Az Egotree-t definiáljuk a következő módon,

  $EgoTree(s, \bar{p}, \Delta)$:

  - s a forrás csomópont
  - $\bar{p}$ a szomszédainak eloszlásai
  - Δ fokszám

- Ez közel optimális megoldást ad torlódásra és úthosszra

1. Tétel. *Adott egy $\bar{p}$ frekvencia eloszlás az s forrás ponthoz, és adott egy Δ fokszám, ekkor az $EgoTree(s, \bar{p}, \Delta)$ egy (α, β)-cl-DAN a következő paraméterekkel:*

- $\alpha = \frac{4}{3}$
- $\beta = log^2(\Delta + 1)$

2.5. $EgoTree(s, \bar{p}, \Delta)$ algoritmus

1. s a gyökér elem, Δ fokszámmal, üres fa
2. Rendezzük sorba $\bar{p} = \{p1, p2, ..., p_k\}$ valószínűségeket csökkenő sorrendben
3. Kezdjük rárakni a fára a csomópontokat, a gyökér elemre legfeljebb Δ levél kerülhet
4. Mikor elértük a Δ levelet, a következő csomópontokat mindig a legkisebb összesített súlyú levélre kapcsolok rá, itt már legfeljebb két levele lehet minden fának

2.5.1. Algoritmus elemzése

- A kapott eredményben látható, hogy a maximális torlódás a legnagyobb súlyú élen van
- Minimalizálni ezt, lényegében egy időzítés probléma, hogy osszuk ki a munkákat Δ processzornak, hogy minden leghamarabb kész legyen
- Erre az optimális algoritmus NP-nehéz, de van közelítő módszer

2.5.2. Longest Processing Time (LPT)

- Először sorba rendezzük a feladatokat hossz szerint csökkenő sorrendben
- Ha van szabad processzor, akkor ahhoz rendeli a leghosszabb munkát
- Ha nincs akkor ahhoz a processzorhoz rendeli, ahol a legkevesebb ideig tart a munka

2. Tétel. *Legyen ω_L a maximum idő, mielőtt egy processzor befejezi az összes munkát a mohó LPT algoritmus szerint, és ω_0 az optimális, ekkor*

$$\frac{\omega_L}{\omega_0} \leq \frac{4}{3} - \frac{1}{3\Delta}$$

Ez az algoritmus polinom időben lefut

1. Lemma. *Az $EgoTree(s, \bar{p}, \Delta)$ ad egy $\frac{4}{3}$ szorzóval nagyobb közelítést a minimális torlódásra az optimális Δ fokú fához képest, ami kiszolgál $\bar{p}$ frekvencia eloszlást egy adott s forrás csomópontra*

2. Lemma. *Az $EgoTree(s, \bar{p}, \Delta)$ ad egy $log^2(\Delta + 1)$ szorzóval nagyobb közelítést a minimális úthosszra az optimális Δ fokú fához képest, ami kiszolgál $\bar{p}$ frekvencia eloszlást egy adott s forrás csomópontra*

2.6. cl-DAN algoritmus

3. Tétel. *Legyen D egy szimmetrikus kommunikáció kéréseloszlás , ahol az átlag csúcs fokszáma ρ, (azaz az élek száma $\rho \cdot \frac{n}{2}$. Ekkor a maximum fokszám $\Delta = 12\rho$, ehhez lehetséges generálni egy (α, β)-cl-DAN hálózatot, ahol:*

- $\alpha = 1 + (\frac{8}{9})\Delta$
- $\beta = 1 + 4log^2(\Delta + 1)$

Konstans ρ esetén ez konstans közelítést ad a minimális torlódásra és az optimális úthosszra

1. Felosszuk a hálózat csúcsait két halmazra, H - magas és L - alacsony fokszámúakra fele-fele arányban
    - Az alacsony fokszámú csúcsok fokszáma legfeljebb 2ρ
2. Megkeressük az összes olyan (u, v) élt, ahol u és v is a magas fokszámú halmazba tartozik
3. Az ilyen éleket a gráfban kiegészítjük egy segítő csomóponttal, $l \in L$, az eredeti csomópontok között megszüntetjük az élt, és felveszünk két új élt (u, l) és (v, l)
    - Minden segítő l csúcs választásakor egy még nem felhasználtat válasszunk az L halmazból
4. Meghatározunk egy mátrixot, ami első lépésben az eredeti
    - Ahol segítő csomópontot vettünk fel, ott az útvonal hosszúhoz hozzá kell még adni az l-el való áthaladást is, és törölni kell az eredeti pontok közti élt
    - Ezután elkészítjük a magas halmaz csúcsaira a T_u fát, ahol a valószínűségeket a mátrixból kiolvassuk, $\Delta = 12\rho$ fokszámmal, ez közel optimális megoldást ad mindkét fel

5. Mivel u és v pontok közt egy l segítő csomópont van használva ezért T_u és T_v módosításra szorul. Alakítsuk át először T_u-t T'_u-ra

   - Ha $l \notin T_u$, $(p(u,l) = 0)$, akkor l átveszi v helyét T'_u-ban
   - Ha $l \in T_u$, $(p(u,l) > 0)$, akkor két lehetőségünk van:
     - Ha $(p(u,l) > (p(u,v))$, akkor töröljük v-t a fából
     - Ha $(p(u,l) \leq (p(u,v))$, akkor l átveszi v helyét T'_u-ban
   - T'_v hasonlóan számítjuk ki, ezzel garantálva, hogy T'_u és T'_v közötti kommunikáció az l csomóponton keresztül fog áthaladni

6. Konstruáljuk meg az új N hálózatot, vegyük az előbb készített egofákat és vegyük az uniójukat, azaz húzzuk be az összes olyan élet amik szerepeltek a fákban

   - De mivel nem csak magas fokú csomópontok közt történhetett adatforgalom, ezért még vegyük hozzá az N hálózathoz azokat az éleket is, ahol mindkét csomópont alacsony fokszámú volt

3. fejezet

Megvalósítás

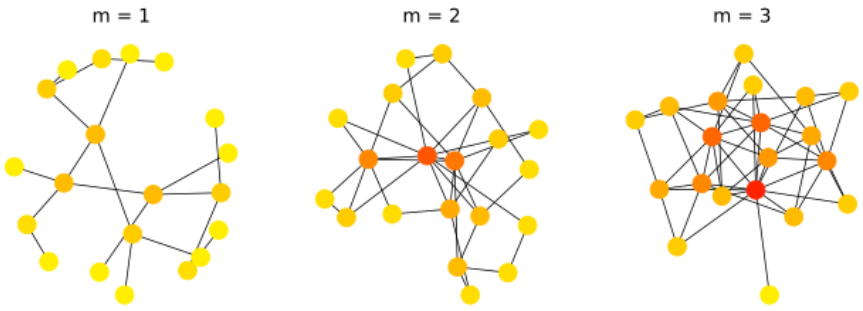
3.1. Keretrendszer

A keretrendszer Python 3 nyelven íródott, és a Networkx külső csomag volt használva a véletlen gráfok generálására. A példakód megtalálható futtatható hagyományos Python programként és Jupyter notebookban [3]. Networkx csomag továbbá biztosít számunkra egy megjelenítési lehetőséget, amit a Jupyter notebookban tudunk legjobban kihasználni. Az adatok feldolgozása és kiértékelése is Jupyter notebookban történt.

3.2. Véletlen gráfok

3.2.1. Barabási-Albert gráf

A Barabási-Albert gráf[7] a komplex hálózatok egy modellje. A gráf felépítése az úgy nevezett "preferential attachment" mechanizmust használja, ami kimondja, hogy az újonnan becsatlakozó csomópontnak m már a hálózatban szereplő csomópontra kell kapcsolódnia. A 3.1 ábrán látható a gráf eredménye, attól függően, hogyan választjuk meg az m értéket.



3.1. ábra. Barabási-Albert gráf

3.3. Modell

A **Network** osztály valósítja meg az algoritmus legnagyobb részét, de magában nem használható, mivel nem tartalmazza a fa építési stratégiát. Ahhoz, hogy a teljes legyen az osztály, le kell származtatni és meg kell valósítani milyen algoritmussal építse meg a fákat. Ennek segítségével nagyon egyszerűen lehet új algoritmust illeszteni a már meglévő rendszerhez. A Network osztály bemenete a demand mátrix, kimenete egy útválasztási séma. Ez mellett sok metaadatot is kiszámol a program amik között szerepel az átlag súlyozott úthossz és a torlódás. Az algoritmus futásához még szükség egy paraméterre, ahol megadjuk a maximális fokszámot a rendszernek, ami alapján elkészülnek a fák. Itt kettő lehetőségünk van, kötött és dinamikus fokszám attól függően mit szeretnénk elérni. A dinamikus fokszám esetén az átlag fokszám függvényében lehet megadni a maximális fokszámot. Erre egy példa a "6d", ahol azt fejezzük ki, hogy az átlag fokszám hatszorosát szeretnénk használni az adott gráfban, mint maximális fokszám a megépítendő fák esetén. Az útválasztási séma létrehozásakor a csere lépéssorozatot követően megváltozik a demand mátrix, ezért fontos, hogy a Δ fokszámot ne haladjuk meg, de ez nem garantált. A metaadatok tartalmazzák a kiszámított maximális delta fokszámot és a valós fokszámot, ami a szükséges fokszámot adja meg amire algoritmusnak szüksége volt. Ebből az adatból lehet következtetést levonni, hogy valóban megfelelő-e felső korlátja amit a szerzők adtak és lehet-e jobb felső korlátot adni.

Az algoritmushoz szükséges bemeneti demand mátrix több módos is megadható konfiguráción keresztül. A konfigurációk egy *JSON* fájl tartalmazza, amiben egyszerre több konfiguráció is megadható. A lehetséges konfigurációk pedig a következők

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

Többféle módon lehet megadni konfigurációt attól függően milyen gráfot akarunk használni. Lehetőség van kézileg megadni a demand mátrixot vagy generálhatunk kétféle gráfot. Véletlen gráfok amit tud generálni a program:

- Erdős-Rényi gráf
- Barabási-Albert gráf

Egy minta konfiguráció, ami tartalmaz példát mind három esetre:

```
{
        "config": [ {
                "graph": "erdos-renyi",
                "vertex_num": 11,
                "dan": null,
                "constant": 3
```

```
        }, {
                "graph": "barabasi-albert",
                "vertex_num": 11,
                "dan": 3,
                "m": 4
        }, {
                "graph": "manual",
                "vertex_num": null,
                "dan": 3,
                "demand": [
                [0, 3, 4, 1, 1, 1, 1],
                [3, 0, 2, 0, 1, 0, 4],
                [4, 2, 0, 2, 0, 0, 4],
                [1, 0, 2, 0, 3, 0, 0],
                [1, 1, 0, 3, 0, 0, 0],
                [1, 0, 0, 0, 0, 0, 3],
                [1, 4, 4, 0, 0, 3, 0]]
        } ]
    }
```

A konfigurációnak van három kötelező mezője, és a egy kiegészítő mező, attól függően, hogy melyik típusú gráfot választottuk.

Kötelező mezők:

1. **graph** - a gráf típusa, három érték közül lehet választani:
   - *"erdos-renyi"* - Erdős-Rényi gráf
   - *"barabasi-albert"* - Barabási-Albert gráf
   - *"manual"* - Kézzel megadott gráf
2. **vertex_num** - A gráf csúcsainak száma. Ez nincs figyelembe véve kézzel adott gráf esetén, mivel meg van adva a demand mátrix és nem kell azt kigenerálni.
3. **dan** - A Demand-Aware Network fokszámára ad megkötést. Értékek amit felvehet:
   - *null* - Alap beállítás szerint a cikkben ajánlott 12ρ fokszámot fogja használni
   - *12* - A megadott szám lesz a fokszám, példában most 12.

- *"6d"* - A szám és egy "d" betűvel a végén hasonlóan működik mint az első eset, annyi különbséggel, hogy itt meg lehet adni, hogy mi legyen a ρ szorzója, példa most 6ρ

Konfiguráció függő mezők:

- **constant** - *Erdő-Rényi* gráf esetén a modell vár egy p valószínűséget ami annak a valószínűsége, hogy egy él be legyen húzva az új pontba, és ez független minden a korábban behúzott élektől. A ritka mátrix eléréséhez a következő formulát használja a generáló algoritmus:

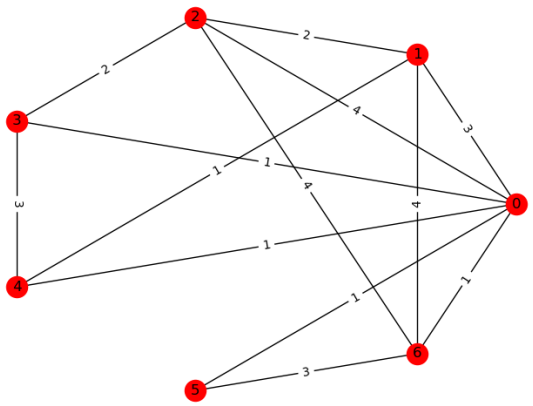
$$p = constant \cdot \frac{1}{vertex\_num}$$

- **m** - *Barabási-Albert* gráf esetén a modell vár egy m számot, ami azt adja meg, hány olyan gráf ponthoz kell csatlakozzon az új, ami már eddig benne van a gráfban.

- **demand** - *kézzel megadott* gráf esetén direkt módon megadható a mátrix, aminek a formája listák listája. Figyeljünk arra, hogy négyzetes lehet csak a demand mátrix!

3.4. Kimenet

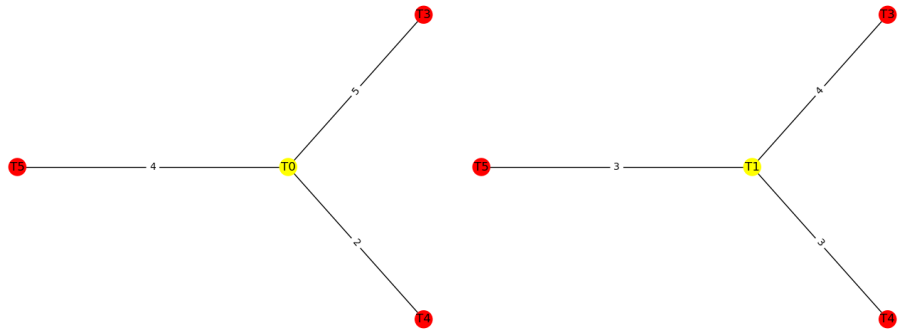
Az program kimenete, az algoritmus által kiszámolt metrikák, átlag súlyozott úthossz és torlódás. Ha a rajzolás opció be van kapcsolva, akkor a kiindulási hálózat, az egófák és az új hálózat választási séma ki lesz rajzolva.



3.2. ábra. Kiindulási hálózat

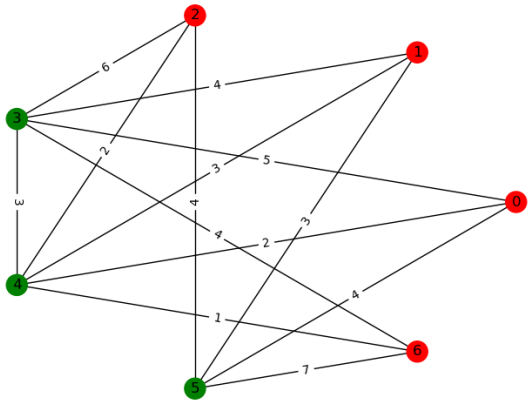
A 3.2 ábrán látható a bemeneti hálózat, aminek a demand mátrixa már a korábban fel lett írva. Az algoritmus ezek után a cl-DAN algoritmust használva, először

elkészíti az egófákat 3.3 ábra, majd végül az új hálózat választási sémát ami a 3.4 ábrán látható.



3.3. ábra. Egófák

A 3.4 ábrán látható gráfon pár extra információ megfigyelhető. Pirosra vannak festve a magas fokszámú csúcsok és zöldre az alacsony fokszámúak. Az algoritmus fő célja az volt, hogy ne legyen egymással közvetlen kapcsolatban két magas fokszámú csúcs, azaz ne legyen két piros csúcs összekötve, és ez maradéktalanul teljesül is. Két magas fokszú csúcs csak egy alacsony fokszámú segítő csúcson keresztül tud kommunikálni. Fokszámok szempontjából a csúcsok rendben vannak, mivel nem haladják meg Δ fokszámot. A gráf esetén a delta $\Delta = 12\rho = 12 \cdot \lceil \frac{25}{7} \rceil = 43$.



3.4. ábra. Új hálózat

3.5. 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

4. fejezet

Bevezetés

4.1. Labor célja

A labor célja a korábban már bemutatott cikkben[9] szereplő hálózat megépítése. Az elkészült keresztrendszert kiegészíteni még egy fa építési stratégiával, ami a Véletlenszerűen felépített fa (Random fa). Továbbá még különböző metrikák bevezetése, ami segítségével pontosabb képet kapunk arról, hogy különböző helyzetekben milyen eredményeket eredményeznek az algoritmusok. Egy fontos megközelítésbeli változás is alkalmazva volt az algoritmus első lépésére, mennyi fát építünk és az milyen kihatással van az eredményre.

4.2. Laborban megvalósított munka

A labor ideje alatt a már elkészült keretrendszer segítségével tesztek voltak futtatva, és azok adati kiértékelve. A keretrendszer Python [6] nyelven íródott. A véletlen gráfok generálására a NetworkX külső csomag volt használva[5]. Az adatok elemzése Python segítségével történt.

5. fejezet

Modell

5.1. EgoBalance

Az EgoBalance 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.

5.2. Huffman fa

A eddigi két algoritmus az Egofát[9] 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[11] 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.

5.3. 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.

5.4. Random fa

Az eddigi fa építési stratégiák mindegyike valamilyen szempontból próbált egy jobb útválasztási sémát létrehozni. A Random fa olyan megközelítést használ, hogy szimulálja azt ha valaki véletlenszerűen kötögetné össze a csomópontokat. Az így létrehozott fák semmit nem használ olyan információt ami hatékonyabb eredményhez vezetne. Az algoritmus a már korábban használt Sorfolytonos fára épít, azaz a csomópontokat sorfolytonosan helyezzük el. Egyetlen változás az algoritmusban, a fához tartozó vektor ami tartalmazza a kommunikációs valószínűségeket nincs sorba rendezve, hanem ötször meg vannak benne keverve az elemek.

5.5. Módosított fa építés

Az cikkben szereplő cl-DAN algoritmus első lépése a csomópontok besorolása magas és alacsony halmazokba a fokszámuk alapján. A szerzők itt fele-fele arányban osztják el a pontokat és egy hozzáadott feltételként megnézik, hogy az alacsony fokszámúak között szerepel-e olyan csomópont, aminek a fokszáma meghaladja 2ρ-t, azaz a kétszeres átlag fokszámot.

A általam módosított cl-DAN algoritmus ezt a kiegészítő szabályt veszi alapul. Mi lenne ha, nem szabályosan fele-fele arányba lenne elosztva, hanem magán a 2ρ feltételen?

Nézzünk egy példát, ahol ez jelentőséggel bír. Tegyük fel, hogy van egy 25 csúcsú csillag gráfunk két csillaggal. A csillag gráfban a csillag közepére kapcsolódik rá az összes másik csomópont, így van kettő 24 fokú csomópontunk, és huszonhárom darab 2 fokú csomópont. Az átlag fokszám így $\rho = 23 \cdot 2 + 2 \cdot 23 = 3.68$, ennek a kétszerese $2\rho = 7.36$.

Az eredeti esetben a nagy fokú csomópontokhoz a csillagok közepei fog tartozni és további tizenegy darab 2 fokú csomópont. Így legalább $\lceil \frac{n}{2} \rceil$ fát fogunk építeni, ahol a pontok nagy része olyan kapcsolatban van, hogy két 2 fokszámú között egy segítő csúcs van, ami hasonló hozzájuk és eredetileg 2 fokszámú volt. Ezzel a módszerrel jelentősen növeltük meg a torlódást a gráfban.

A módosított algoritmus az eredetivel ellenben a magas halmazban csak két darab csúcsot tartalmaz, a középpontokat. Ezen esetben ténylegesen magas csomópontok közé helyezünk egy kisegítőt, ami bármelyik lehet a maradék 2 fokszámú csúcsok közül, ezzel redukáltuk is az épített fák számát.

A módosított algoritmus nem lesz rosszabb mint az eredeti algoritmus, mivel ott mit kiegészítő feltétel szerepel a fokszám ellenőrzés, még itt az alapot adja, így legrosszabb esetben is visszatérünk az eredetihez.

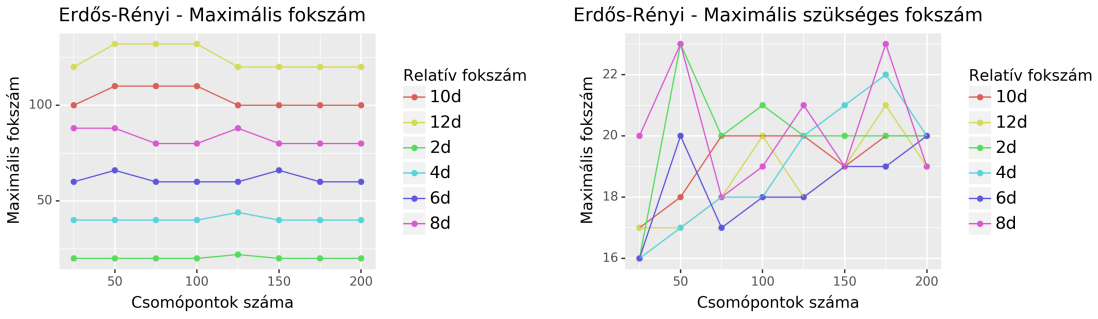
6. fejezet

Tesztelés

6.1. Tesztelés menete

A megfelelő teszt mennyiség eléréséhez véletlen gráfok lettek generálva. Mind a három gráf típus lett tesztelésre használva, ezek paramétere pedig változott egy és tíz között.

A különböző kapacitás szimulálására a maximum fokszám meg lett határozva, hogy a következő halmaz elemit veheti fel $\Delta \in [10, 16, 24, 48]$. A tesztkörnyezet támogat az átlag fokszám függvényében megadott Δ értéket. A tesztek segítségével ki lett mutatva, hogy egy ponton túl, már annyira nagy lesz ez az érték, hogy minden csúcs a gyökérre kapcsolódik. Ezzel teljesen értelmét vesztve, hogy milyen fa építési stratégia volt használva, mivel a fa nem éri azt az elemszámot, hogy legyen nem direkt csatlakozó csomópont a gyökérhez. A 6.1 ábrán látható ez:



6.1. ábra. Erdős-Rényi gráf - Fa mennyiség összehasonlítás

A következő szempont amiben változtak a gráfok, hogy mekkora terhelés legyen az éleken. Itt két csoportba lehet sorolni a teszteket, ahol egytől tízig véletlenszerűen volt kiválasztva, a második esetben pedig minden él egységesen egy terhelést kapott.

Következő szempont amit figyelve volt az maga az algoritmus változtatása, hogy mennyi fa készüljön el, és hogy milyen stratégiával.

Végül pedig, hogy a mérési hiba minimalizálásának érdekében az fent említett paraméterek összes kombinációjára húsz teszt futott le. Összességében 384.000 teszt készült le, amit meg lehet találni a projekt GitHub oldalán.

6.2. Metrikák

Tesztek különböző metrikák alapján lettek kiértékelve, amit a következők:

- `graph` - gráf típusa
- `vertex num` - csúcsok száma a gráfban
- `constant` - a gráfhoz tartozó konstans paraméter
- `congestion` - torlódás az eredeti értékekkel
- `real congestion` - a torlódás normalizálva egyre
- `avg route len` - átlag úthossz
- `delta` - fa Δ fokszáma
- `max delta` - a maximális Δ fokszám
- `dan` - a Δ megadott fokszám, ami lehet relatív is
- `most congested route` - a legnagyobb torlódással rendelkező él
- `max route len` - a maximális úthossz
- `avg tree weight` - az átlag fa súlya
- `most tree ratio` - a legnagyobb arány fa átlag ágsúlya és a legnehezebb ág között
- `tree count` - az épített fák száma
- `type` - a fa építései algoritmus típusa
- `start entropy` - a kezdeti költség mátrix entrópiája

7. fejezet

Teszt eredmények kiértékelése

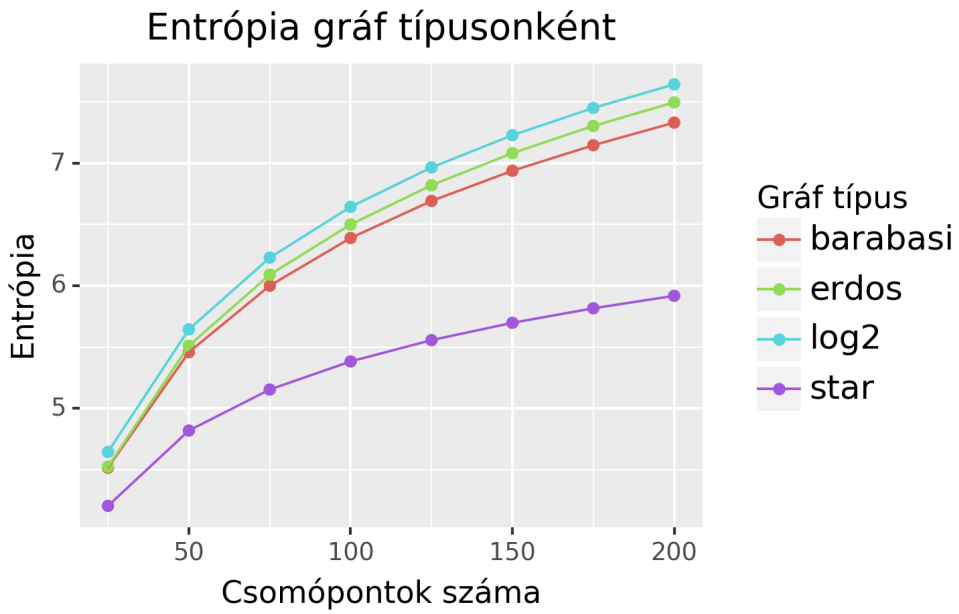
7.1. Entrópia

A különböző gráf típusok különböző entrópiával rendelkeznek. Az entrópia diszkrét valószínűségi változóra felírható a következő képlettel[8]:

$$H(X) = \sum_{i=1}^{n} p(x_i) \cdot log_2 \frac{1}{p(x_i)};\ \forall i \in [1, ..., n] \wedge x_i = \sum_{1}^{n} demaind\_matix[i][n]$$

Megjegyzés: mikor $0 \cdot log_2 \frac{1}{0}$ értéket vesz fel az x_i változó, akkor legyen $x_i = 0$. Legyen $\bar{p}$ a demand mátrix, ekkor $H(\bar{p})$ megegyezik a következővel $H(p_1, p_2, ..., p_n)$, ahol p_i a mátrix egy sorában szereplő valószínűségek összege. Ha teljesül, hogy $p_i > 0$ és $\sum_i p_i = 1$ és a $\bar{p}$ egyenletes eloszlást követ, akkor a véletlen gráfban az entrópiára felső korlátja $H(\bar{p}) = \log n$, ahol n a csomópontok száma.

A fenti képlet segítségével ki lehet számolni az entrópiát a különböző véletlen gráfokra. A következő grafikon mutatja, hogy a tesztek során használt véletlen gráfoknak mennyi az entrópiájuk. Eredmény a 7.1 ábrán.



7.1. ábra. Entrópia

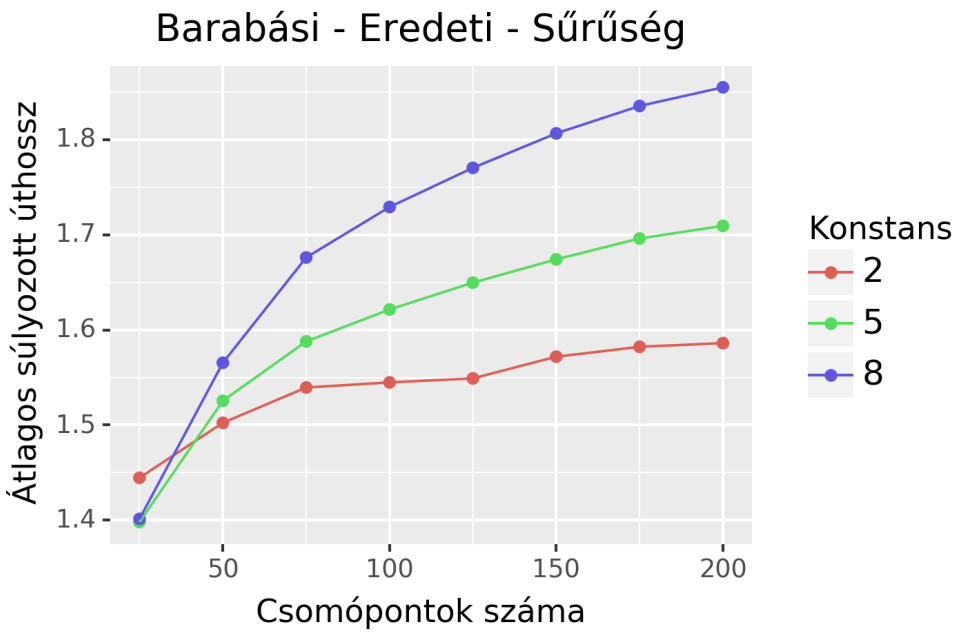
A grafikonon látható, hogy az Erdős-Rényi gráfnak van a legnagyobb entrópiája, itt egy valószínűség változó határozza meg mennyi éle lesz a gráfnak. Ezt követi a Barabási-Albert gráf, ahol tudjuk mennyi élt várunk, annak függvényében mennyi régi csomópontra kell kapcsolódnia az új csomópontnak. Majd végül a legkisebb entrópiát a csillag eredményezte, mivel a csúcsok csak a csillag középpontokhoz csatlakoznak, máshova nem.

7.2. Úthossz

7.2.1. Általános eset

A tesztek során a konstans érték határozta meg, hogy mennyire volt kitöltve a demand mátrix. Ezért először nézzük meg mennyire van kihatással a mátrix kitöltöttsége az eredményekre.

Eredmény a 7.2 ábrán.



7.2. ábra. Átlagos súlyozott úthossz és demand mátrix kitöltöttségének kapcsolata

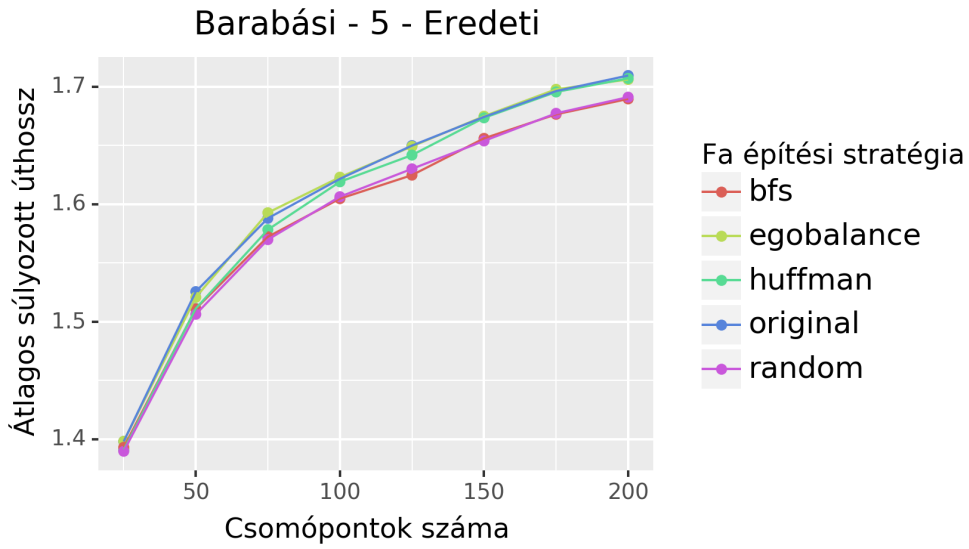
A grafikonon a Barabási-Albert gráf eredményei látható, az eredeti mennyiségű fa megépítésével, azaz legalább minden második csúcsponthoz elkészült, és a fák pedig az eredeti algoritmussal készültek el. A hálózatban az összes szereplő él súlya 1, normalizálás után pedig $\frac{1}{|E|}$. Mint látható a grafikonon, minél ritkább a mátrix, annál rövidebbek az úthosszak is. A további grafikonoknál már csak a konstans 5 értékű eredményeket fogom vizsgálni, mivel az ad egy jó közelítést az átlagra.

7.2.2. A fa építő algoritmusok összehasonlítása

Eredeti megépített fa mennyiség

Az általános esetben csak egy véletlen gráfnak a konkrét esetét néztük meg, most vizsgáljuk meg, hogy a különböző fa építési stratégiák, hogy befolyásolják az úthosszt.

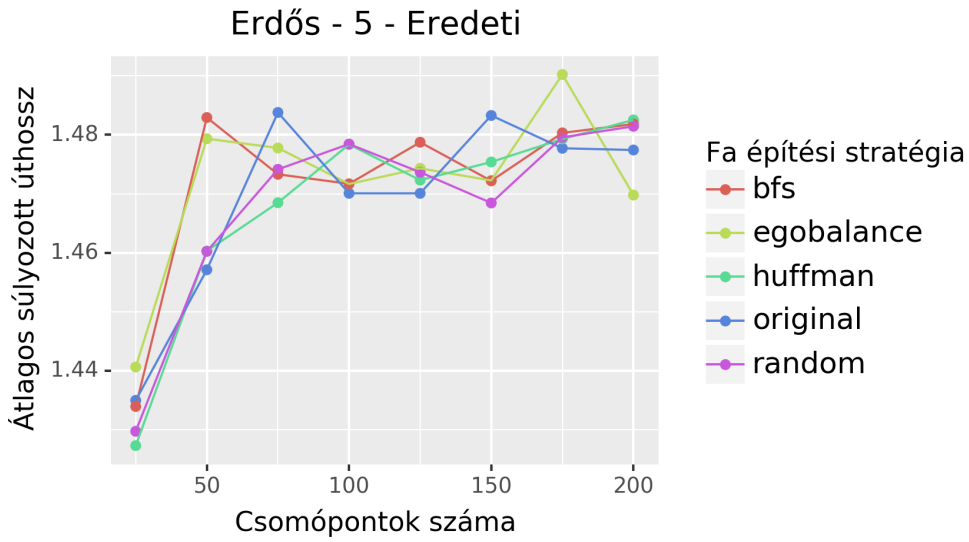
Első véletlen gráf típus a Barabási-Albert gráf, ahol az eredeti számú fát építettük meg. Eredmény a 7.3 ábrán.



7.3. ábra. Barabási-Albert gráf - Úthossz

A grafikonon látható, hogy három csoportba lehet besorolni az algoritmusokat. Az elsőbe tartozik az EgoBalance és az Eredeti algoritmus. Ezek rendelkeznek a legnagyobb átlagos úthosszal és szinte ugyanazt az eredmény adják, mérési hiba különbséggel. A következő a sorban a Huffman fa alapú algoritmus, ami kezdetben alacsonyabbról indul, de végül csatlakozik az első kettőhöz. Legjobb két algoritmus pedig a Sorfolytonos fa és a Random fa. Ez várható volt, mivel mindkettő ugyanazt az algoritmust használja, csak az értékekben különböznek. Úthossz szempontjából ez a kettő adja mindig legkisebb fát, mivel egy teljes fát épít az algoritmus.

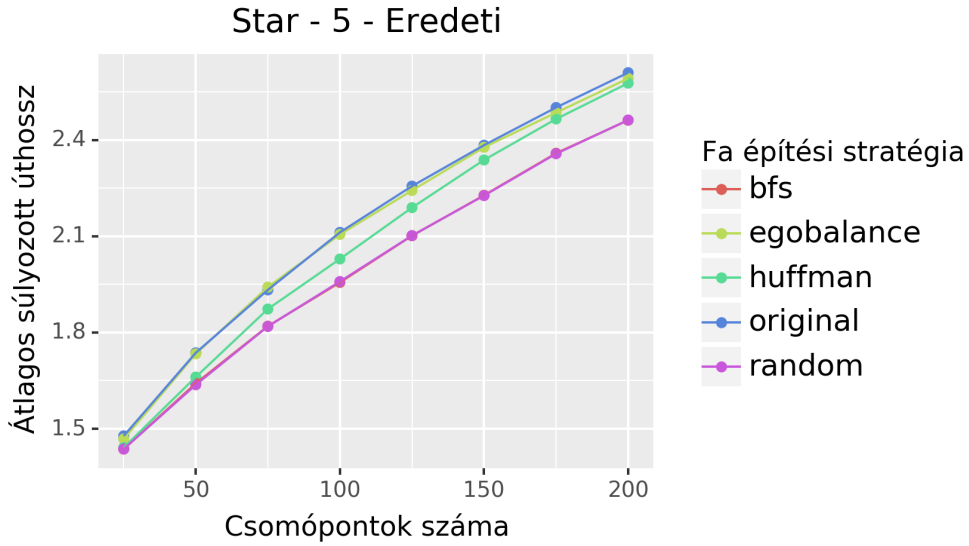
Következő véletlen gráf típus amit vizsgálunk az az Erdős-Rényi gráf, itt is az eredeti számú fát építjük meg. Eredmény a 7.4 ábrán.



7.4. ábra. Erdős-Rényi gráf - Úthossz

A grafikon itt már kicsit érdekesebb, mivel kicsit nagyobb mozgása van az értékeknek, de ha megnézzük, legfeljebb kettő százalékos eltérés mérhető. Az összes algoritmus hasonlóan teljesít.

Végül pedig nézzük meg a csillag gráfot az eredeti számú fa mennyiséggel a 7.5 ábrán.



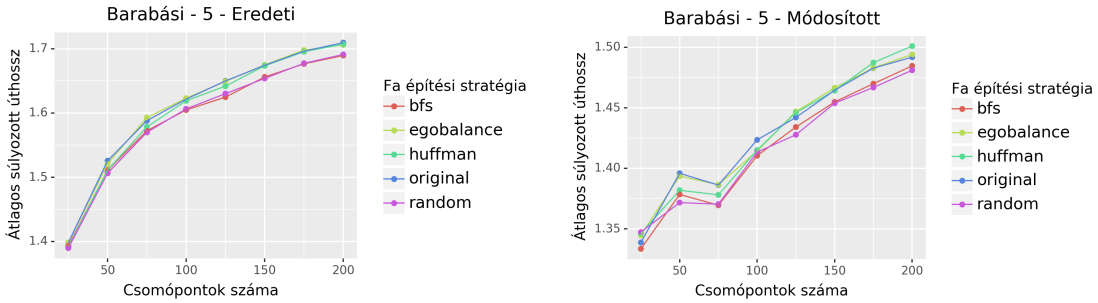
7.5. ábra. Csillag gráf - Úthossz

Mint már láttuk a Barabási-Albert gráf esetében, itt is megvan a három egyértelmű kategória, ami teljesen megegyezik az előzővel.

Módosított megépített fa mennyiség

Az előző részben láthattuk milyen eredményeket adnak az eredeti feltétel alapján a különböző algoritmusaink. Most nézzük meg, hogyan változik az úthossz a megépített fák számának függvényében. Azokat a fákat építjük meg amit ténylegesen nagyfokú, teljesül a magas fokszámú pontokra, hogy a fokszámuk legalább kétszerese az átlag fokszámnak.

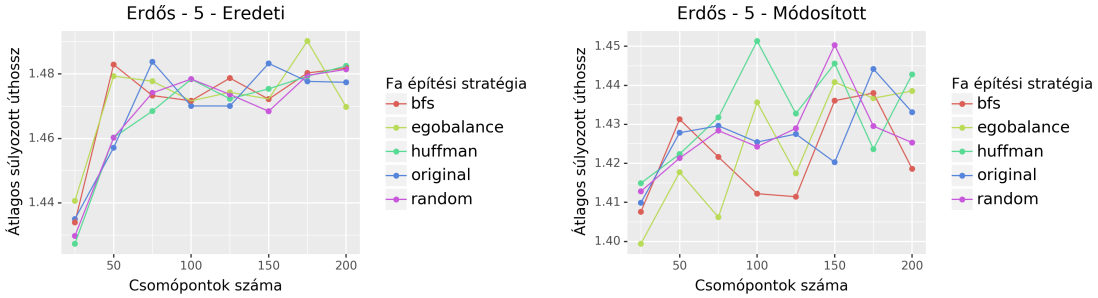
Hasonlítsuk össze a Barabási-Albert gráf eredményeit a 7.6 ábrán.



7.6. ábra. Barabási-Albert gráf - Megépített fa számosságának összehasonlítása

Első jelentős különbség, hogy csökken az átlag úthossz. Az gráf iránya továbbra is tartja az eredeti trendjét, egy kis beeséssel a 75 csúcsú gráfnál, de látható, hogy javít a módosítás az eredeti algoritmushoz képest.

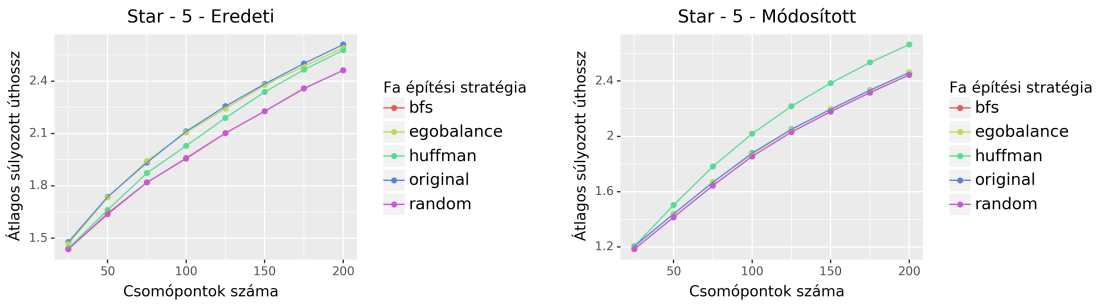
Következő gráfunk az Erdős-Rényi gráf a 7.7 ábrán.



7.7. ábra. Erdős-Rényi gráf - Megépített fa számosságának összehasonlítása

Amint látható, a grafikonon ismét megjelenik egy szórás, kicsit nagyobb is mint az eredetinél, de még mindig a pár százalékos határon belül. Az eredetihez képest az értékek kisebbek, ezzel elérve célunk a módosítás bevezetésével.

Végül nézzük meg a csillag gráfot a 7.8 ábrán.



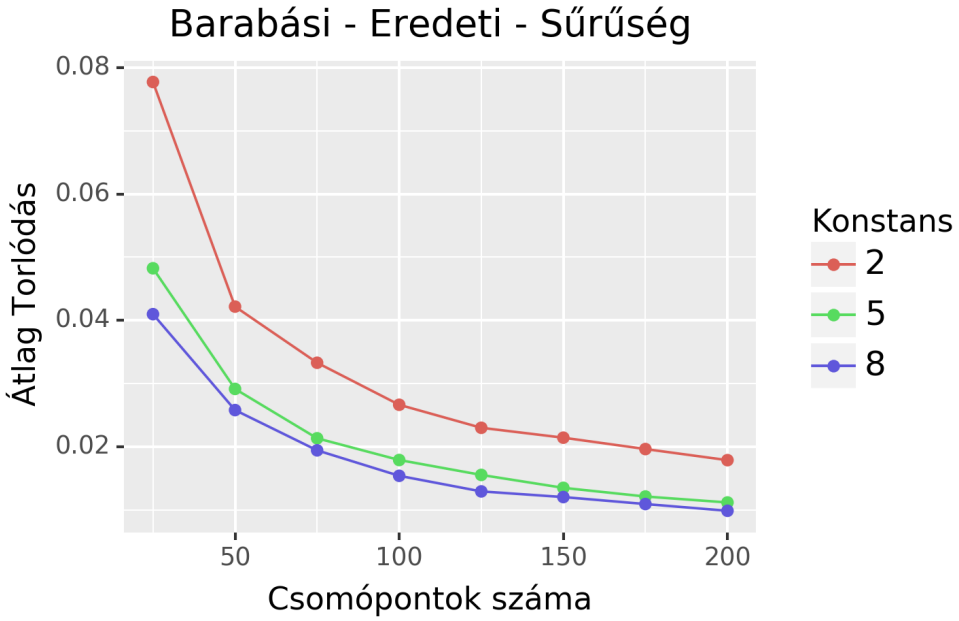
7.8. ábra. Csillag gráf - Megépített fa számosságának összehasonlítása

Itt történt egy fordulat a grafikonon, először is amíg a Huffman fa alapú stratégia a közép értéket adta a két másik csoport között, itt most jelentősen rosszabb eredmény produkált. A másik négy algoritmus meg javult az úthosszara nézve és megmaradt a relatív pozíciójuk.

7.3. Torlódás

7.3.1. Általános eset

Az úthosszhoz hasonlóan először nézzük meg, hogy az eredeti algoritmus milyen eredményt ad, attól függően mennyire sűrű a gráf. Eredmény a 7.9 ábrán.



7.9. ábra. Torlódás és demand mátrix kitöltöttségének kapcsolata

A grafikonon a Barabási-Albert gráf eredményei látható, az eredeti számú meg-

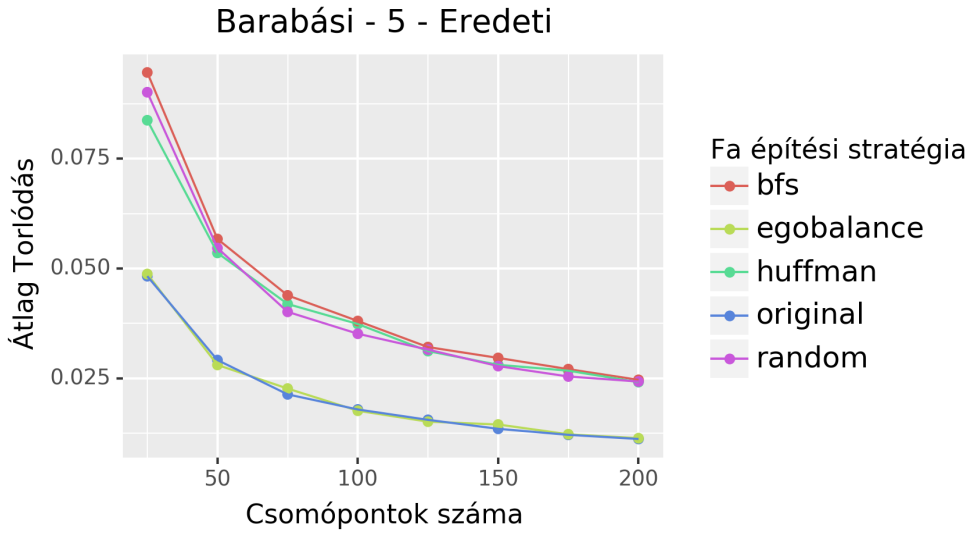
épített fa mennyiséggel és a fák pedig az eredeti algoritmussal készültek el. A hálózatban az összes szereplő él súlya 1. Mint látható a grafikonon, minél ritkább a mátrix, annál nagyobb a torlódás. Egy fontos észrevétel a két metrika között, még az úthossz átlagosan az egész mátrixra nézve adta meg az eredményt, addig a torlódás az egyértelműen a legnagyobb torlódás az útválasztási sémán. Ezért ha kevés éllel rendelkezik a gráf, annál kevesebb lehetősége van olyan élt választani az algoritmusnak, ahol még alacsony a torlódás. A további grafikonoknál már csak a konstans 5 értékű eredményeket fogom vizsgálni, mivel az ad egy jó közelítést az átlagos torlódása.

7.3.2. A fa építő algoritmusok összehasonlítása

Eredeti megépített fa mennyiség

Az általános eset után, most vizsgáljuk meg, hogy a különböző fa építési stratégiák, hogy befolyásolják a torlódást.

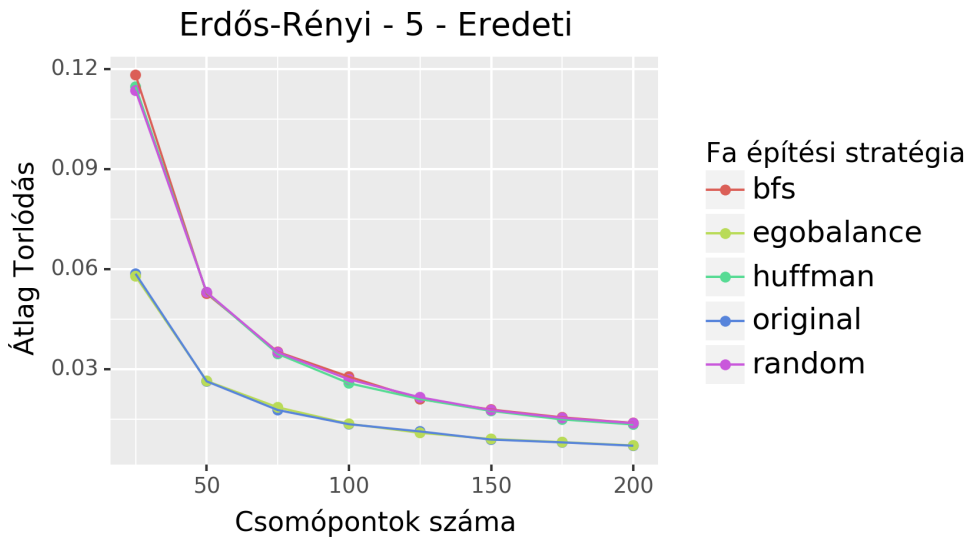
Első véletlen gráf ismét a Barabási-Albert gráf, ahol az eredeti mennyiségű fát építjük meg. Eredmény a 7.10 ábrán.



7.10. ábra. Barabási-Albert gráf - Torlódás

A grafikonon látható, hogy két csoportba lehet besorolni az algoritmusokat. Az elsőbe tartozik az EgoBalance és az Eredeti algoritmus. Ezek adják a legjobb eredményt és szinte azonosak. A másik csoportba tartozik a maradék három algoritmus, a Sorfolytonos-, a Random- és a Huffman fa. Itt egyértelmű miért jött ki ez az eredmény, mivel ez a három algoritmus egyáltalán nem veszi figyelembe a tényezőt, hogy mekkora a torlódás.

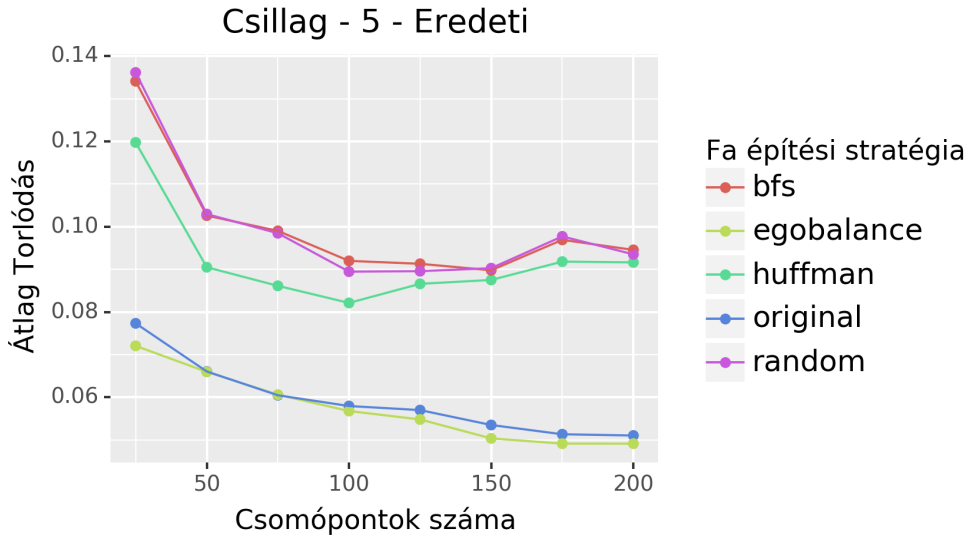
Következő véletlen gráf típus az Erdős-Rényi gráf, itt is az eredeti számú fát építjük meg. Eredmény a 7.11 ábrán.



7.11. ábra. Erdős-Rényi gráf - Torlódás

A grafikon szinte megegyezően ugyanazt az eredmény mutatja mint a Barabási-Albert gráf esetén. Két csoport, ahol még mindig az Eredeti és az EgoBalance teljesítenek a legjobban.

Végül pedig nézzük meg a csillag gráfot az eredeti fa mennyiséggel a 7.12 ábrán.



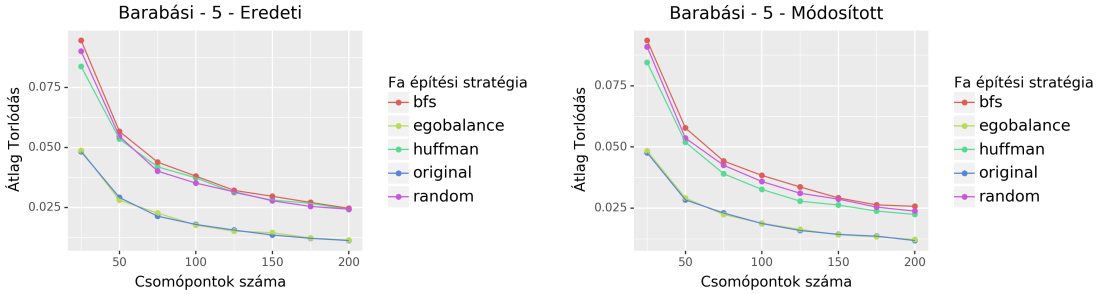
7.12. ábra. Csillag gráf - Torlódás

Itt már elhatárolódik a Huffman fa a Sorfolytonos és Random fáktól, de nem eléggé, hogy megközelítse az Eredetit vagy az EgoBalance-ot.

Módosított megépített fa mennyiség

Az előző részben láthattuk milyen eredményeket adnak az eredeti feltétel alapján a különböző algoritmusaink. Most nézzük meg, ha változik a torlódás a megépített fák mennyiségének függvényében. Azokat a fákat építjük meg amit ténylegesen nagyfokúak.

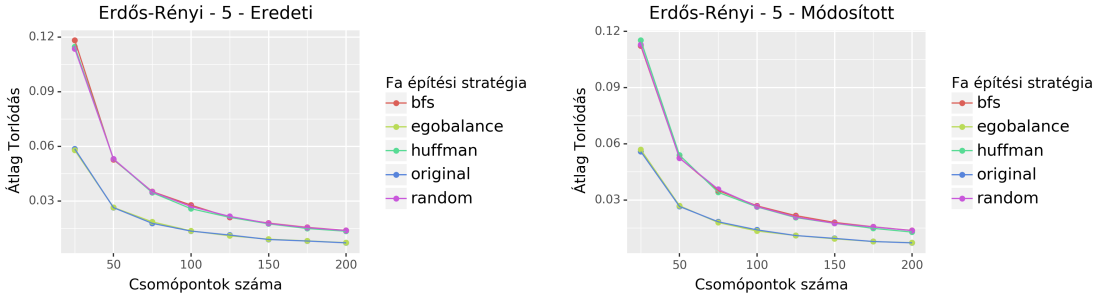
Hasonlítsuk össze a Barabási-Albert gráf eredményeit a 7.13 ábrán.



7.13. ábra. Barabási-Albert gráf - Fa számosság összehasonlítás

Első jelentős különbség nem jelentkezik.

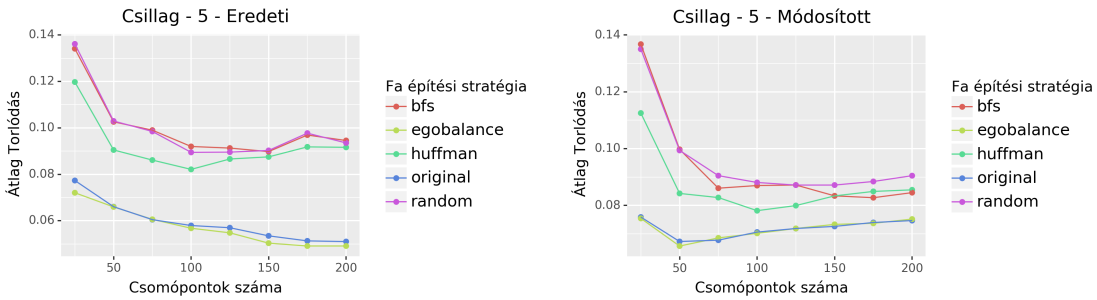
Következő gráfunk az Erdős-Rényi gráf a 7.14 ábrán.



7.14. ábra. Erdős-Rényi gráf - Fa számosság összehasonlítás

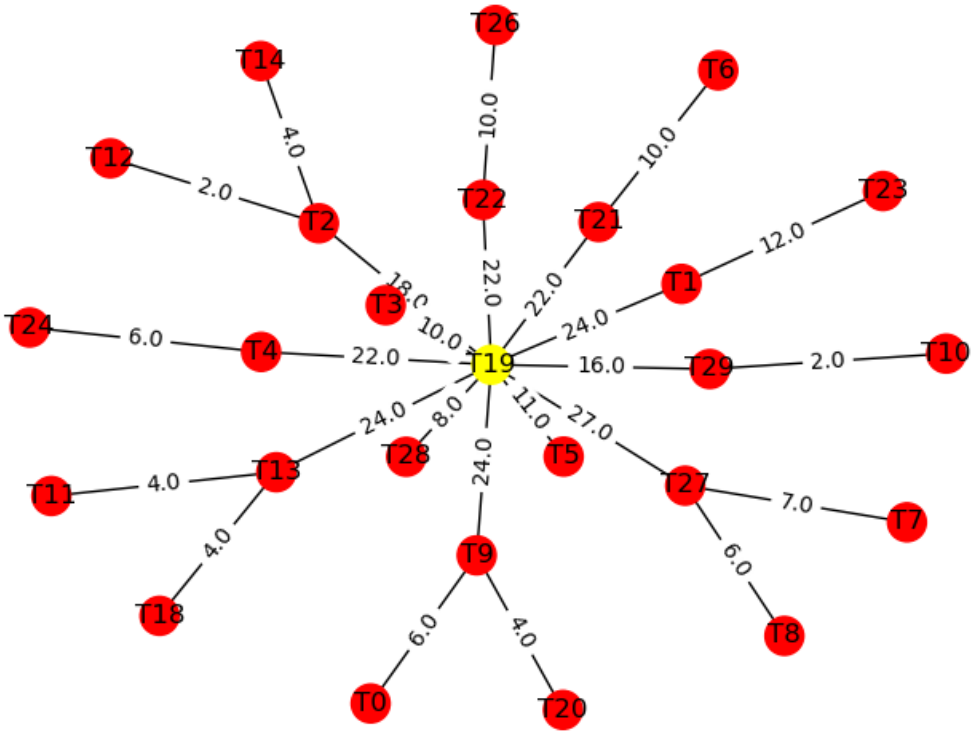
Jelentős különbség itt sem figyelhető meg.

Végül nézzük meg a csillag gráfot a 7.15 ábrán.



7.15. ábra. Csillag gráf - Fa számosság összehasonlítás

Itt már jelentkezik különbség, az elején még hasonló a két gráf, de ahogy növeljük a csomópontok számát, a módosított algoritmus rosszabb eredményt eredményez, mint az eredeti algoritmus. Ennek az az oka, hogy a megépült fák sekélyek, de ahogy növekedik a élek száma, egyre nagyobbak lesznek ezek a fák. Szóval a torlódás összpontosul egy helyre, erre pedig példa a 7.16 ábra.



7.16. ábra. Csillag gráf - Egofa az eredeti algoritmus alapján

A hálózat ami ezt az Egofát eredményezte az 30 csomópontból áll, 5 csillagot tartalmaz és maximális Δ fokszám pedig 12. Mint látható egységesen helyezkednek el a csomópontok és a legtávolabbi levél is csak kettő mélységre helyezkedik el. Abban az esetben, mikor a csomópontokból relatív kevés van, és alacsony fák kapcsolódnak

a torlódás is várhatóan alacsony lesz. Ahogy nő a csomópontok száma és eléri a többszörösét a maximális fokszámnak, nehéz utak alakulnak ki és így növekszik a torlódás is.

8. fejezet

Összefoglalás

8.1. Labor eredménye

8.1.1. Random fa algoritmus

A Random fa algoritmusa nem lett a legjobb algoritmus az összes közül, de egy érdekes tényre mutatott rá. A teljes fa építése mindig rövidebb utat eredményez, mintha különböző hosszúságú ágak lennének a fában torlódástól függően. Ezért ez az algoritmus mindig jobb eredményt adott úthosszra, mint a cikkben megfogalmazott. Mikor torlódáshoz értük, akkor egyértelműen látszik, hogy a véletlen szerűen összekapcsolt pontok nem lesznek torlódás optimálisak. Az algoritmus alapját a Sorfolytonos fa adta, ezért mindkettő algoritmus szinte ekvivalens eredményhez vezetett.

8.1.2. A megépített fák hatása az eredményre

A labor lényegi kérdés az volt, hogy ha csak tényleg a szükséges fákat építjük meg akkor jobb lesz-e a hálózatunk. A kapott eredményekből pedig az látszik, igen jobb lesz az úthossz esetében, mivel az algoritmus mindig rövidebb úthosszt eredményez, mint az eredeti. Az torlódás már kicsit érdekesebb eredményt mutat, mivel a torlódásra nincs feltétlen kihatással, ha egy kiegyensúlyozott gráfot nézünk. Ellenben, ha ez nem teljesül, hanem egy extrém esetet veszünk, a csillag gráfot, ott egyenesen rosszabb lesz az eredmény.

8.2. A munka eredménye

A [9] cikk által meghatározott algoritmus megfelel arra, hogy teljesítse a célját, közel optimális útválasztási sémát készítsen egy ritka forgalmi mátrixszal reprezentált hálózathoz. Attól függően, hogy értelmezzük a cikk egyes részeit, mikor csere lé-

pést alkalmazzuk, kaphatunk két algoritmust, az Eredetit vagy az EgoBalance-t. A kettő közötti különbség többnyire mérési hiba, ezért mindkettő elfogadható. A másik három algoritmus is hasonló eredménnyel végzett, ami közül a Sorfolytonos fa egy befutó, ha a rövid utakat akarunk elérni minden áron. Az megépített fák mennyiség is egy fontos tényező, mivel jelentős úthossz csökkenéshez vezet, tolódás szempontjából pedig csak elfajult esetekben lesz rosszabb mint az eredeti algoritmus.

9. fejezet

Irodalomjegyzék

[1] Cisco Data Center Infrastructure 2.5 Design Guide - Data Center Architecture Overview [Design Zone for Data Center Networking].

[2] Do you know the data center network architecture? | Optcore.net.

[3] Jupyter - https://jupyter.org/.

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

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

[6] Python - Python.org.

[7] R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97, Jan 2002.

[8] C. Avin, K. Mondal, and S. Schmid. Demand-aware network designs of bounded degree. *CoRR*, abs/1705.06024, 2017.

[9] 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.

[10] M. Ghobadi, D. Kilper, R. Mahajan, A. Phanishayee, N. Devanur, J. Kulkarni, G. Ranade, P.-A. Blanche, H. Rastegarfar, and M. Glick. ProjecToR: Agile Reconfigurable Data Center Interconnect. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference - SIGCOMM '16*, pages 216–229, Florianopolis, Brazil, 2016. ACM Press.

[11] 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.