Eötvös Loránd Tudományegyetem

Informatikai Kar

Információs Rendszerek Tanszék

Forgalom igény tudatos hálózat tervezés

Témavezető:
Lukovszki Tamás
Egyetemi docens, PhD
Információs Rendszerek tanszék

Készítette:
Szecsődi Imre
Programtervező informatikus
MSc

Budapest, 2019

Tartalomjegyzék

1. Bevezetés **3**
1.1. Motiváció . . . 3
1.1.1. Valós forgalmi adatok . . . 4
1.1.2. Hálózat tervezési stratégiák . . . 4
1.1.3. Adatközpontok hálózati felépítése . . . 5
1.1.4. Újrakonfigurálás megvalósítása . . . 6
1.2. Diplomamunka célja . . . 6
1.3. Diplomamunka eredménye . . . 7
1.4. Diplomamunka felépítése . . . 7

2. Modell **9**
2.1. Forgalom igény tudatos hálózat tervezés probléma . . . 9
2.2. Formális definíció . . . 9
2.2.1. Torlódás . . . 10
2.2.2. Úthossz . . . 10
2.2.3. Skálázhatóság . . . 10
2.2.4. Optimális torlódás . . . 11
2.2.5. Optimális úthossz . . . 11
2.3. cl-DAN hálózat tervezése . . . 11
2.4. EgoTree . . . 11
2.4.1. EgoTree algoritmus . . . 12
2.4.2. Algoritmus elemzése . . . 13
2.4.3. Longest Processing Time (LPT) . . . 13
2.5. cl-DAN algoritmus . . . 13

3. Kiterjesztett modell **15**
3.1. Véletlen gráfok . . . 15
3.1.1. Barabási-Albert modell . . . 15
3.1.2. Erdős-Rényi modell . . . 16
3.1.3. Csillaggráf . . . 16
3.2. Fa építési stratégiák . . . 17

3.2.1. EgoBalance . . . 17
3.2.2. Huffman fa . . . 18
3.2.3. Sorfolytonos fa . . . 20
3.2.4. Random fa . . . 21
3.3. Módosított fa építés . . . 22

4. Megvalósítás **23**
4.1. Keretrendszer . . . 23
4.2. Adatszerkezetek . . . 23
4.3. Hálózat modell . . . 24
4.4. Kimenet . . . 26

5. Tesztelés **29**
5.1. Tesztelés menete . . . 29
5.2. Metrikák . . . 30

6. Teszt eredmények kiértékelése **31**
6.1. Entrópia . . . 31
6.2. Úthossz . . . 32
6.2.1. Általános eset . . . 32
6.2.2. A fa építő algoritmusok összehasonlítása . . . 33
6.3. Torlódás . . . 37
6.3.1. Általános eset . . . 37
6.3.2. A fa építő algoritmusok összehasonlítása . . . 38

7. Összefoglalás **42**
7.1. Diplomamunka eredménye . . . 42
7.2. Keretrendszer . . . 42
7.3. Algoritmusok . . . 43
7.4. Magas fokszámú csomópontok . . . 44

8. Köszönetnyilvánítás **45**

9. Irodalomjegyzék **46**

1. fejezet

Bevezetés

A diplomamunka a Demand-Aware Network Design with Minimal Congestion and Route Lengths [11] cikk alapján készült.

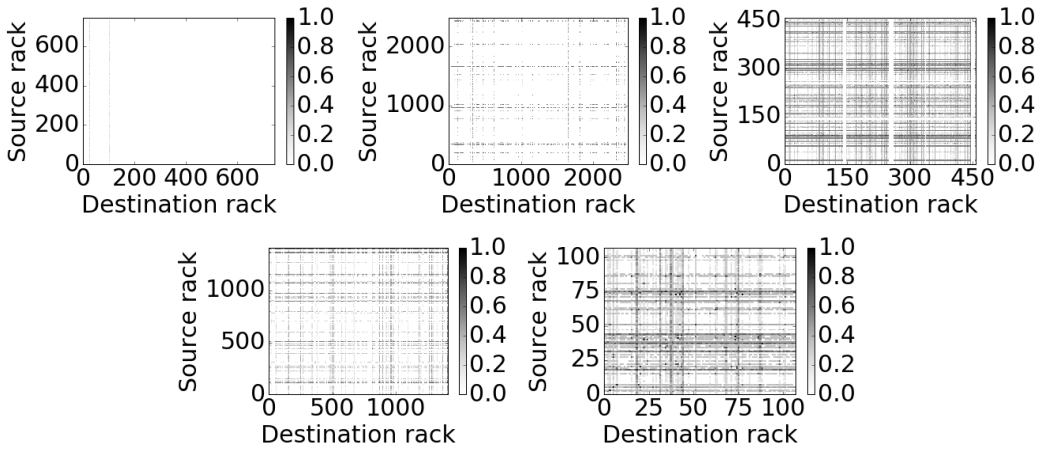
1.1. Motiváció

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 két szerver. A gyakorlati tapasztalatok azt mutatják, hogy a kommunikáció túlnyomó része néhány szerver között történik. Mikor a kommunikációs mintát ábrázoljuk egy gráfban, ahol az élek az egymással kommunikáló 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 [13], ami lézer technológia segítségével köti össze a szervereket egy adatközpontban. Az új technológiák új problémákat eredményeztek, és új kihívások elé állították a hálózat tervezést. Napjainkban intenzív kutatás folyik ezen a területen.

1.1.1. Valós forgalmi adatok

A Microsoft Research ProjecToR kutatás keretén belül a szerzők adatokat gyűjtöttek Microsoft adatközpontokban. Adatokat kétszázötvenezer szerverről rögzítettek, amelyek öt productionban használt klaszterben voltak elosztva.



1.1. ábra. Microsoft adatközpont adatok klaszterenként [13]

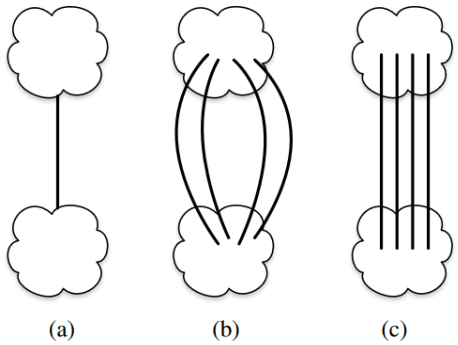
Amint az 1.1 ábrán látható, a kommunikáció főleg megadott szerverek között történik. Korábban már említve volt, az adatközpontok jelenleg a legrosszabb esetre vannak tervezve, hogy bármelyik két szerver tudjon kommunikálni. Az ábrán látszik, hogy melyik szerverek között nem szükséges feltétlen direkt 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 adatközpontok milyen topológiával rendelkeznek jelenleg.

1.1.2. Hálózat tervezési stratégiák

A technika fejlődésével elérhetővé váltak eszközök arra, hogy egy adott hálózatot újrakonfiguráljunk, attól függően, hogy 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 adatközpontban milyen forgalomeloszlást rögzítettek, és ez alapján lehet majd újratervezni a hálózatot, figyelve a terheléseloszlásra. Lehetőség szerint, periodikus újrakonfigurálással akár még nagyobb hatékonyságot is elérhetünk az adatközpontokban.

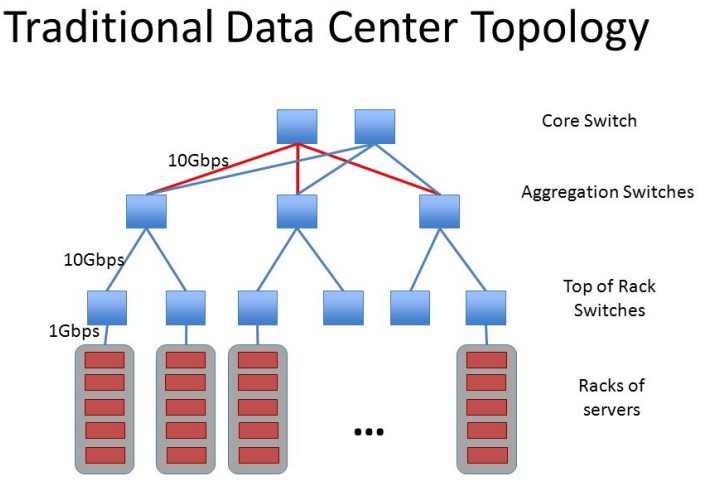
Két fő optimalizációs megközelítést fogunk megnézni a munka során, ezek a rövid úthossz és a minimális torlódás. Rövid úthossz alatt azt értjük, hogy minél kevesebb ponton kelljen áthaladnia az adatnak mielőtt az eljut a céljába, 1.2 ábrán az (a) eset. A minimális torlódás alatt pedig azt értjük, ha 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 az 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 [11]

1.1.3. Adatközpontok hálózati felépítése



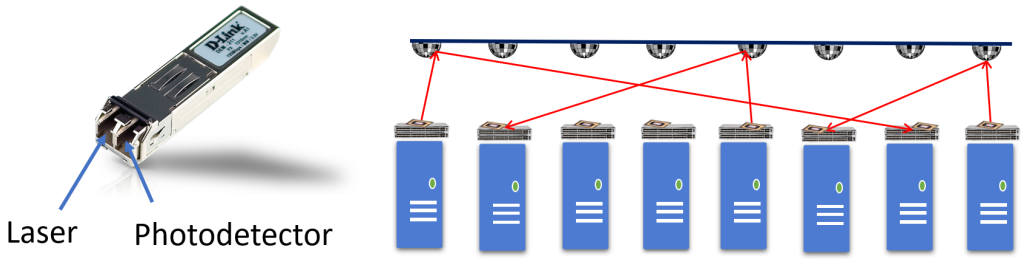
1.3. ábra. Adatközpont hálózati topológiája

Mielőtt még bármi nemű átalakítást végeznénk az adatközpont hálózatán, először 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 adatközpontból. A Core réteg további szerepe még, hogy adatot szolgáltasson az alatta lévő Aggregációs rétegnek. Ez a réteg nagyobb csoportokra osztja az adatközpontot, ezzel minimalizálva a routing táblák számát. Az

aggregációs réteg további szolgáltatásokat is tud nyújtani az alatta lévő rétegnek, mint például tűzfal, 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 kiépítve 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. Az 1.4 ábrán látható eszköz szem számára láthatatlan lézer nyalábbal küld és fogad adatokat. Az adatközpont 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. A váltásidő itt $12\mu s$, ami kétszázötvenezerszer gyorsabb, mint az optikai switch.



1.4. ábra. ProjecToR [13]

1.2. Diplomamunka célja

A diplomamunka célja a cikkben [11] bemutatott algoritmus implementálása. Ezt egy olyan környezetben megvalósítani, ahol empirikus módszerekkel tesztelhető az algoritmus hatékonysága. Ez mellet a keretrendszer könnyen kiegészíthető legyen tesztelés szempontjából, továbbá lehetőséget biztosítson arra, hogy új algoritmust

lehessen bevezetni, és azt hasonlóan megvizsgálni, mint az eredetit. A megvalósítás így több független modulból áll, amelyek egyként fognak működni. Az így kapott összes eredményt összehasonlítjuk a cikkben megadott elméleti korlátokkal.

1.3. Diplomamunka eredménye

A diplomamunka véletlen gráfokkal modellezte az adatközpontokban kialakulható hálózati forgalmat. Az eredmények azt mutatják, hogy a cikkben meghatározott "optimális" eset egy határozottan magasabb felső korlát, mint amire valójában szükség van. A cikkben felvázolt algoritmus némi helyet hagy az megvalósítónak, mivel nem teljesen írja le, hogy pár helyzetben mi a helyes lépés.

Ez alapján született két algoritmus, az Eredeti és az EgoBalance, ami az egyik ilyen rész miatt más-más értelmezésben próbálta megoldani a felmerülő helyeztet. A teljesség érdekében egy másik megközelítési módszer is vizsgálva volt, mint fa építési stratégia, ami további három algoritmust eredményezett, ezek rendere a Huffman-, Sorfolyotonos- és Véletlen fák voltak. Ezek az algoritmusok az úthosszt vették figyelembe, és itt jól is teljesítettek, de torlódás szempontjából alul múlták a cikkben szereplőt közel egy kettes szorzóval. Ebből kifolyólag a cikkben szereplő algoritmus megfelelő a céljára, de nem zárja ki a lehetőséget egy másfajta javítási megközelítésre.

A második javítási mód előtérbe helyezi azt a szempontot, hogy mérlegeljük a hálózatban szereplő ténylegesen leterhelt pontokat, és azokat építjük át csak. Ezzel elejét vehetjük az olyan esteknek, ahol nincs szükség átépítésre. Így csökkenthetjük a számítás igényét az algoritmusnak és egyúttal még rövidebb utakat is érünk el az új hálózatban. Ez a javítás nincs kihatással a torlódásra az esetek túlnyomó részében, de vannak elfajult esetek, ahol rosszabbat eredményez.

1.4. Diplomamunka felépítése

A diplomamunka felépítése több részre van bontva.

- Először ismertetésre kerül az eredeti cikkben [11] publikált algoritmus és a hozzá tartozó szükséges háttérismeret.
- A második részben szó fog esni a három véletlen gráfról, ami adta az alapot a tesztelésre.
- Ezt folytatva kitérünk négy darab új algoritmusra, ami valamilyen szempontból változtat a fa építési stratégián annak érdekében, hogy jobb eredményt érjünk el.

- A fa építési algoritmusok után, megvizsgáljuk azt az esetet, hogy mi történik, ha már a legelső lépésben módosítjuk az algoritmust, ezzel mérlegelve azt, hogy mi kerüljön valóban átépítésre a hálózatban.
- Végül következik a tesztelés, és annak kiértékelése, hogy milyen teljesítményt nyújtanak a különböző algoritmusok a különböző véletlen gráfok esetén.

2. fejezet

Modell

Diplomamunkában a [11] cikkben leírt modellt és definíciókat használjuk.

2.1. Forgalom igény tudatos hálózat tervezés probléma

Egy hálózat forgalomigényének nevezzük azt a mátrixot, ami leírja melyik csomópontok állnak kapcsolatban és mekkora valószínűséggel kommunikálnak. Ez a mátrix legtöbb esetben szimmetrikus, de lehetnek esetek egy oldali kommunikációra is, például egy archiváló szerver, ami nem küld vissza információt a küldőjének.

Forgalom igény tudatos hálózat tervezés problémának nevezzük azt, mikor rendelkezésre áll egy meghatározott számú csomóponttal rendelkező hálózat és egy mátrix, ami leírja a kommunikációs mintát. A mátrix elemei súlyozott valószínűségek, hogy i forrásból mekkora eséllyel lesz adat küldve j célba. A cél, hogy ezen adatból egy olyan hálózati sémát készítsünk, ami minimális torlódást és rövid utakat eredményezzen, és ez mellett még skálázható is legyen nagy hálózatokra.

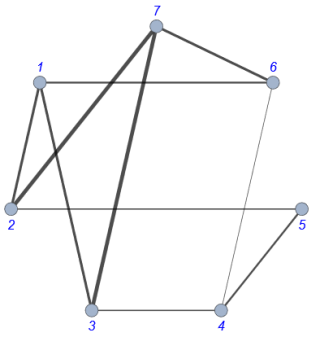
A 2.1 ábrán látható egy demand mátrix és a két megközelítés, ami alapján vizsgáljuk az új hálózatot.

2.2. Formális definíció

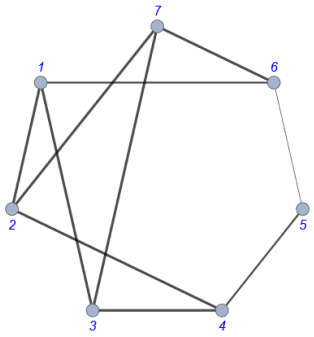
Adott N darab csúcspont $V = \{1, ..., N\}$ és egy M_D kommunikációs séma, ami egy $N \times N$ mátrix. A mátrix (i, j) eleméhez tartozik egy $p(i, j)$ valószínűség, ahol i a forrás, j pedig cél csomópont. A bemeneti mátrix ábrázolható egy irányított G_D gráfban, ahol az élsúlyok a két pont közötti kommunikációs valószínűséget írják le. Az algoritmus feltétele, hogy a M_D mátrix ritka legyen. Egy N hálózatra a torlódást és az úthosszt az útválasztási sémával fogjuk definiálni. Egy útválasztási séma az

	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 [11]

N hálózatra a $\Gamma(N)$, 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:*

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

Szavakba foglalva, a torlódás megegyezik az útválasztási sémában szereplő legjobban leterhelt úttal.

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.

Szavakba foglalva, az átlag súlyozott úthossz, az összes út összege, ahol a csomópontok közötti utak hossza meg van szorozva a valószínűségükkel.

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 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 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, ami a következő:

- *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 α' és β' olyan tényezők amik függetlenek a problémától.

2.4. EgoTree

Az Egófa egy torlódásra és úthosszra optimalizált fa hálózat egy csomópontra nézve. Az Egófát a következő módon definiáljuk:

4. Definició. *Az $EgoTree(s, \bar{p}, \Delta)$ egy fa hálózat, ahol*

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

Ez közel optimális megoldást ad torlódásra és úthosszra.

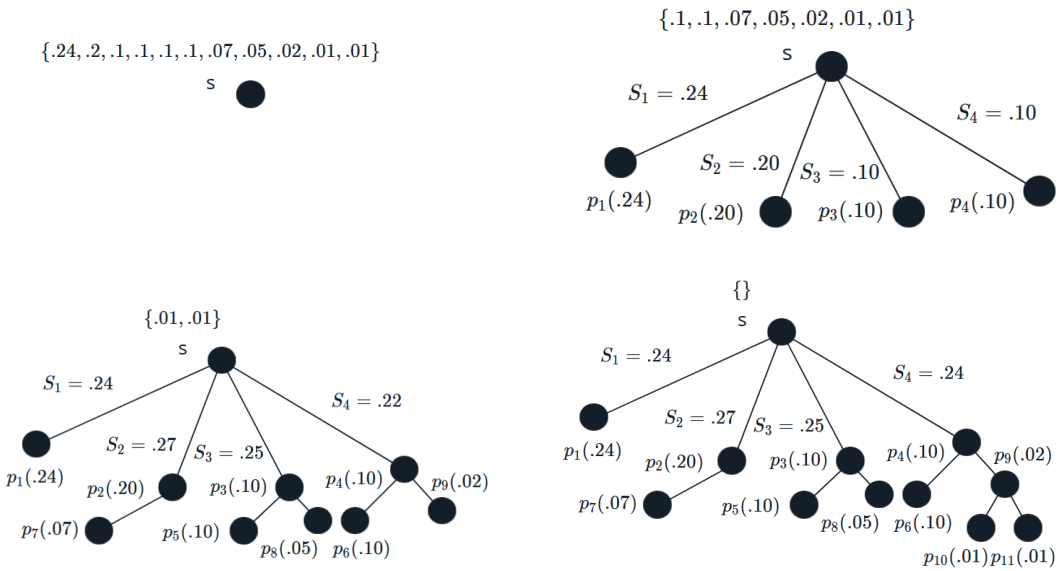
1. Tétel ([11]). *Adott egy $\bar{p}$ frekvencia eloszlás az s forrás ponthoz, és adott egy Δ fokszám, ekkor az $EgoTree(s, \bar{p}, \Delta)$ egy (α, β)-cl-DAN a következő paraméterekkel:*

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

2.4.1. EgoTree algoritmus

Az Egófa megépítésére a következő algoritmust használjuk:

1. s a gyökér elem, Δ fokszámmal, kezdetben üres fa.
2. Rendezzük csökkenő sorrendbe a $\bar{p} = \{p1, p2, ..., p_k\}$ valószínűségeket.
3. Kezdjük a gyökér elemre csatolni a csomópontokat, legfeljebb Δ kapcsolat állhat fent a gyökérrel.
4. Mikor elértük a Δ kapcsolatot a gyökér elemnél, a következő csomópontokat mindig a legkisebb összesített súlyú levélre kapcsoljuk rá, itt már legfeljebb két levele lehet minden fának.



2.2. ábra. Egófa algoritmus lépéseinek vizualizációja a következő valószínűségekre: $\{.24, .2, .1, .1, .1, .1, .07, .05, .02, .01, .01\}$

2.4.2. Algoritmus elemzése

A kapott eredményben látható, hogy a maximális torlódás a legnagyobb súlyú élen van. Minimalizálása a tolódásnak megfeleltethető egy időzítés problémának, ahol gyökérre kapcsolódó csúcsok lesznek a processzorok. Minimalizálás probléma, hogy hogyan osszuk ki a munkákat Δ processzornak úgy, hogy a munka a leghamarabb kész legyen. Erre az optimális algoritmus NP-nehéz [16], de van közelítő módszer.

2.4.3. Longest Processing Time (LPT)

A Longest Processing Time [14] algoritmus a következő:

- Először rendezzük sorba a feladatokat hossz szerint csökkenő sorrendbe.
- Ha van szabad processzor, akkor rendeljük hozzá a leghosszabb munkát.
- Ha nincs szabad processzor, akkor ahhoz a processzorhoz rendeljük hozzá a következő munkát, ahol a legkevesebb ideig tart a munka.

2. Tétel ([11]). *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.*

2.5. cl-DAN algoritmus

3. Tétel ([11]). *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 hosszához 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 megközelítésre
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 egófákat 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

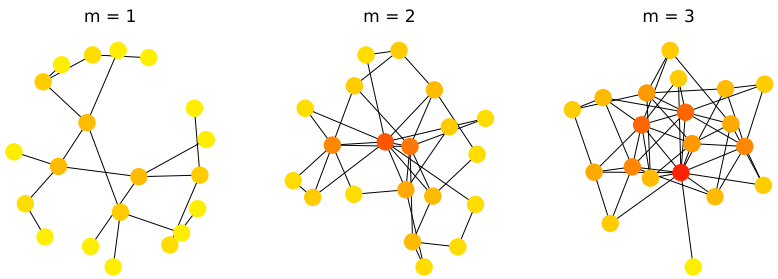
Kiterjesztett modell

3.1. Véletlen gráfok

A hálózat tesztelésének céljából több különböző típusú véletlen gráfon lett tesztelve az algoritmus hatékonysága. A következő alfejezetekben ezek lesznek bemutatva.

3.1.1. Barabási-Albert modell

A Barabási-Albert modell [9] a komplex hálózatok egy modellje. A $G(n, m)$ 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.



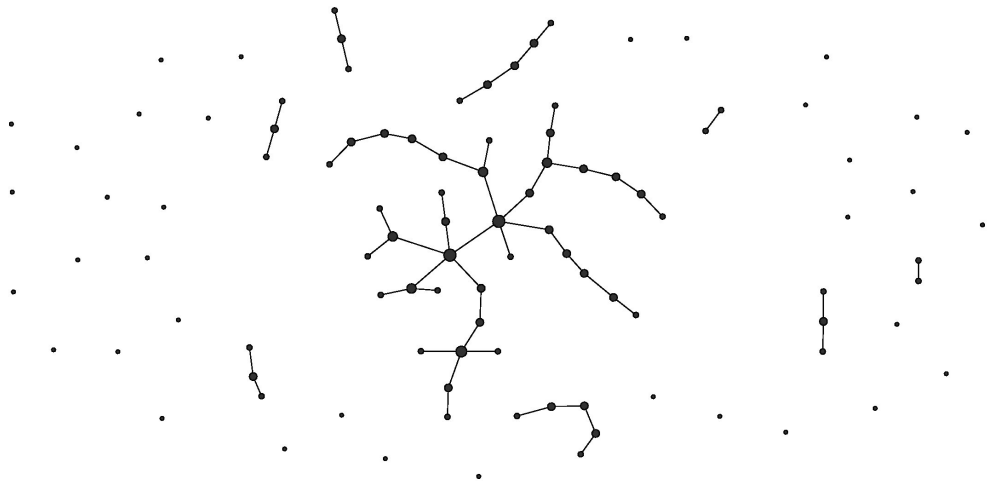
3.1. ábra. Barabási-Albert gráf [6]

3.1.2. Erdős-Rényi modell

Az Erdős-Rényi modell [12] a véletlen gráfok előállítására szolgáló modell. A modellnek két változata van, amik szorosan összefüggenek.

- A $G(n, M)$ modellben egyenletes eloszlás szerint választunk egyet az összes lehetséges n csúcsú gráf közül, ahol az élek száma M.
- A $G(n, p)$ modellben az n csúcsú gráf éleit p valószínűséggel húzunk be bármely két csúcs között, egymástól függetlenül.

A 3.2 ábrán látható egy lehetséges véletlen gráf $p = 0.01$ valószínűséggel.

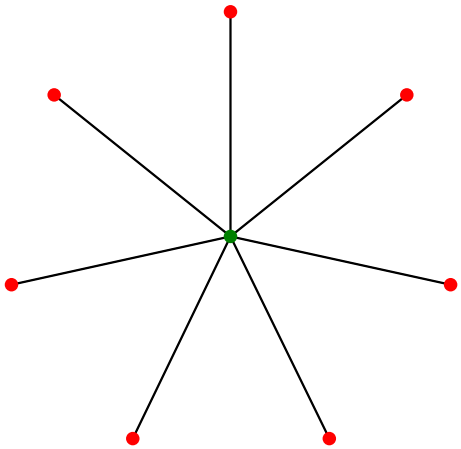


3.2. ábra. Erdős-Rényi gráf $p = 0.01$ valószínűséggel [8]

A munka során a második megközelítés volt használva a véletlen gráfok generálására.

3.1.3. Csillaggráf

A csillaggráf vagy röviden csillag, egy olyan gráf, ahol n csúcs esetén van egy központi csúcs a gráfban, amivel a maradék $n - 1$ csúcs kapcsolatban áll. Bármely két csúcs, amelyek közül egyik sem a csillag közepe, nem áll kapcsolatban. A 3.3 ábrán látható egy példa a nyolc csúcs gráfra, ahol egy csillag van. A munka során használt véletlen csillaggráfokban a csillagok számát lehet megadni.



3.3. ábra. Csillaggráf hét fokszámmal, S_7 [7]

3.2. Fa építési stratégiák

A cikkben [11] már szereplő Egófa algoritmusából kiindulva több különböző fa építési stratégia is meg lett valósítva. A különböző fa építési stratégiák más-más szempontból közelítik meg a problémát.

3.2.1. EgoBalance

Az EgoBalance algoritmus majdnem teljes mértékben megegyezik az eredetivel. A cikk szerzői a cl-Dan 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 lejjebbi szintre kerülni. Ezt a folyamatot addig ismételjük, amíg minden levél nem kerül egy megfelelő helyre.

Az Egobalance algoritmus a függő pontok újra elhelyezését az Egófára 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.

EgoBalance algoritmus

1. Végezzük el a csere lépésig a cl-Dan algoritmust
2. Mikor csere lépés történik a szülő nélküli csúcsokat kapcsoljuk vissza a fába az Egófa algoritmussal

3.2.2. Huffman fa

Az eddigi két algoritmus az Egófát [11] 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 [15] 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 valószínűségeket, és kettesével vonjuk össze őket, amíg nem kapunk egy teljes fát. Mint az Egófáknál, 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:

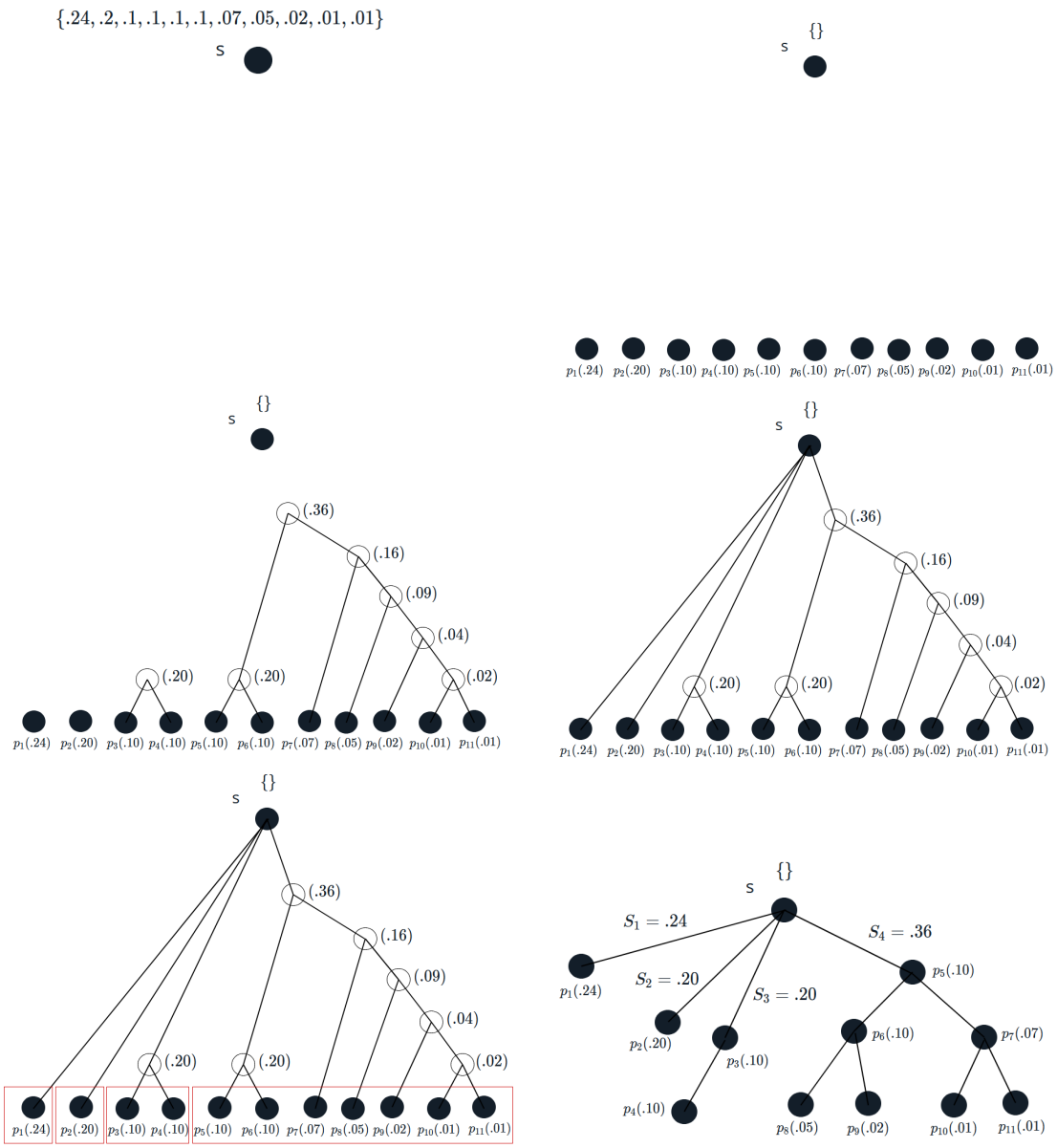
1. Gyűjtsük össze a levelet az áganként, mivel a levelek tartalmazzák a számunkra hasznos pontokat
2. Az ág gyökerétől indulva sorfolytonosan helyezzük el a leveleket, ahol a valószínűségek csökkenő sorrendben vannak rendezve

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 a fa egyik belső pontja sem marad kitöltetlen.

Huffman fa algoritmus

- Végezzük el a fa építés lépésig a cl-Dan algoritmust, demand mátrix itt már a megváltozott állapotban van, nincs két magas csomópont között direkt kapcsolat
- Építsük meg a fákat a következő módon:
  1. Vegyük a gyökérhez kapcsolódó csúcsok valószínűségét egy vektorba
  2. Rendezzük a vektort növekvő sorrendbe
  3. Vonjuk össze a vektor legelső két elemét, majd helyezzük vissza a vektorba és rendezzük újra növekvő sorrendbe
  4. Ezt addig csináljuk amíg az elemek száma nem éri el a maximális fokszámot, Δ-át
- Itt megépítettük a Huffman fát, ahol még nincsenek kitöltve a belső csomópontok
- A köztes csomópontok kitöltése a következő módon történik:
  1. A gyökérre kapcsolódó ágakon gyűjtsük össze az összes levelet
  2. Rendezzük sorba a leveleket nehézség szerint csökkenő sorrendbe
  3. Építsünk teljes fát az ágon



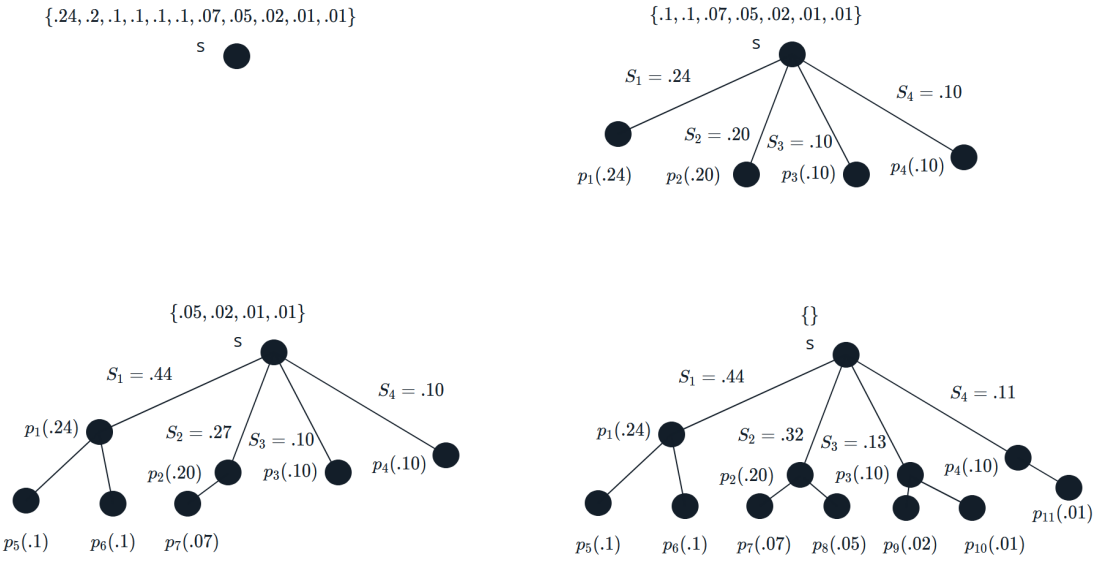
3.4. ábra. Huffman fa algoritmus lépéseinek vizualizációja a következő valószínűségekre: {.24, .2, .1, .1, .1, .1, .07, .05, .02, .01, .01}

3.2.3. Sorfolytonos fa

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

Sorfolytonos fa algoritmus

- Végezzük el a fa építés lépésig a cl-Dan algoritmust, demand mátrix itt már a megváltozott állapotban van, nincs két magas csomópont között direkt kapcsolat
- Építsük meg a fákat a következő módon:
  1. Vegyük a gyökérhez kapcsolódó csúcsok valószínűségét egy vektorba
  2. Rendezzük a vektort csökkenő sorrendbe
  3. Építsünk teljes fát, ahol Δ csomópont kapcsolódik a gyökér elemre



3.5. ábra. Sorfolytonos algoritmus lépéseinek vizualizációja a következő valószínűségekre: $\{.24, .2, .1, .1, .1, .1, .07, .05, .02, .01, .01\}$

3.2.4. Random fa

Az eddigi fa építési stratégiák mindegyike valamilyen szempontból próbált egy jobb fát eredményezni. A Random fa olyan megközelítést használ, hogy szimulálja azt, ha valaki véletlenszerűen kötögetné össze a csomópontokat. Az így létrehozott fa nem használ semmi 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, ami nincs sorba rendezve, hanem ötször meg vannak benne keverve az elemek.

Random fa algoritmus

- Végezzük el a fa építés lépésig a cl-Dan algoritmust, demand mátrix itt már a megváltozott állapotban van, nincs két magas csomópont között direkt kapcsolat
- Építsük meg a fákat a következő módon:
  1. Vegyük a gyökérhez kapcsolódó csúcsok valószínűségét egy vektorba
  2. Keverjük össze a vektor elemeit
  3. Építsünk teljes fát, ahol Δ csomópont kapcsolódik a gyökér elemre

3.3. Módosított fa építés

A cikkben [11] 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.

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

Nézzünk egy példát, ahol ez jelentőséggel bír. Tegyük fel, hogy van egy 25 csúcsú csillaggráfunk két csillaggal. A csillaggráfban 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 fognak 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 mint kiegészítő feltétel szerepel a fokszám ellenőrzés, míg itt az alapot adja, így legrosszabb esetben is visszatérünk az eredetihez.

4. fejezet

Megvalósítás

4.1. Keretrendszer

A keretrendszer Python 3 [5] nyelven íródott és a Networkx [4] 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.

4.2. 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, ami 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 Egófa, 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

4.3. Hálózat modell

A **Network** osztály valósítja meg az algoritmus legnagyobb részét, de magában nem használható, mivel nem tartalmazza a fa építési stratégiát. Ahhoz, hogy 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. Az algoritmus futtatásához szükség van még egy paraméterre, ahol megadjuk a maximális fokszámot a rendszernek, ami alapján elkészülnek a fák. Itt két 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. Az algoritmus kimenete mellett további metaadatok tartalmazzák a kiszámított maximális Δ fokszámot és a valós fokszámot. A valós fokszám adja azt a fokszámot, amire az algoritmusnak tényleg szüksége volt. Ezekből az adatokból lehet következtetést levonni, hogy valóban megfelelő-e a felső korlátja az algoritmusnak, amit a szerzők adtak és lehetőséget ad egy jobb felső korlát becslésre.

Az algoritmushoz szükséges bemeneti demand mátrix több módon is megadható konfiguráción keresztül. A konfigurációt egy *JSON* fájl tartalmazza, amiben egyszerre több konfiguráció is megadható. A lehetséges konfigurációk pedig a következő módon adhatók meg.

Először is, van lehetőségünk egy konkrét mátrixra elkészíteni az útválasztási sémát. Egy így megadott helyes konfiguráció pedig a következő:

```
{
        "config": [
        {
                "graph": "manual",
                "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]]
        } ]
}
```

Fontos paraméterek az ilyen konfiguráció esetén a következők:

- *graph* - itt adjuk meg a bemeneti gráf típusát, esetünkben ez ”*manual*”, azaz ezt a megadott mátrixot fogja használni az algoritmus.
- *dan* - ez a paraméter határozza meg a maximum fokszámot a magas fokú csomópontokra, itt megadható konkrét érték pl. 3, dinamikus érték pl. ”6d” vagy *null* ami ekvivalens a ”12d” értékkel, ami a cikkben [11] meghatározott felső korlát.
- *demand* - maga a demand mátrix, listák listájaként egy négyzetes mátrix, ahol a diagonális elemek értéke 0, mivel önmagával való kommunikáció nem befolyásolja a hálózatot

A konkrét demand mátrix megadás mellett van még lehetőség generálni véletlen gráfot. A program három fajta véletlen gráf típust tud generálni, amik rendre: Barabási-Albert gráf, Erdős-Rényi gráf és csillaggráf.

A véletlen gráfok konfigurációit meg lehet adni hasonló módon mint az előző esetben. Több konfiguráció megadása a $config$ tömbben vesszővel elválasztva történik. A véletlen gráfok konfigurációja megegyezik, és a gráf típusától függően változik a konstans jelentése. Egy példa erre a következő kód részlet:

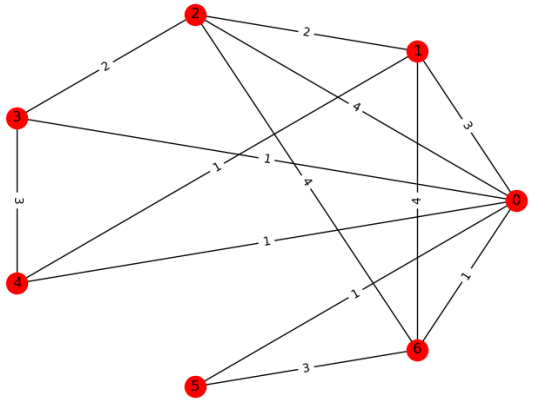
```
{
        "graph": "star",
        "vertex_num": 25,
        "constant": 1,
        "dan": 10
}
```

A paraméterek rendre a következők:

- *graph* - véletlen gráf típusa, aminek a lehetséges értékei:
  - ”*barabasai* – *albert*” - Barabási-Albert gráf
  - ”*erdos* – *renyi*” - Erdős-Rényi gráf
  - ”*star*” - Csillaggráf
- *vertex_num* - Csomópontok száma
- *dan* - A maximális fokszám a magas fokú csúcsokra, megegyezik az előző esettel
- *constant* - Konstans érték ami a megadott gráf paraméterét adja, ezek rendre a következők:
  - Barabási-Albert gráf esetén a "preferential attachment" számát adja meg
  - Erdős-Rényi gráf esetén az él valószínűségét határozza meg a következő képlet segítségével:
    $$p = \frac{constant}{vertex\_num}$$
  - Csillaggráf esetén pedig a csillagok számát adjuk meg

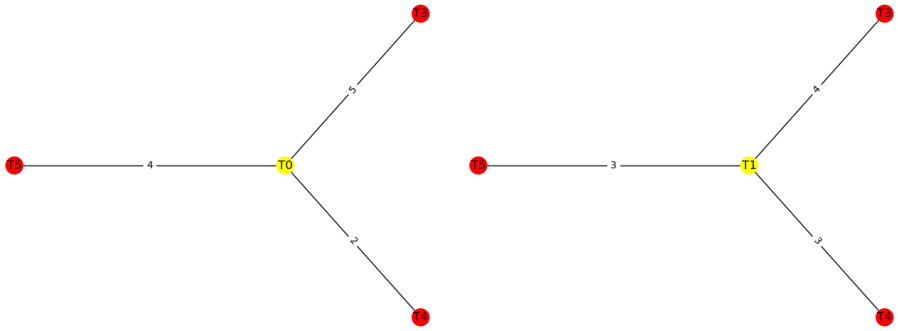
4.4. Kimenet

A program kimenete az algoritmus által kiszámolt metrikák, köztük az átlag súlyozott úthossz és a torlódás. Ha a rajzolás opció be van kapcsolva, akkor a kiindulási hálózat, az egófák és az új hálózat útválasztási sémája lesz kirajzolva.



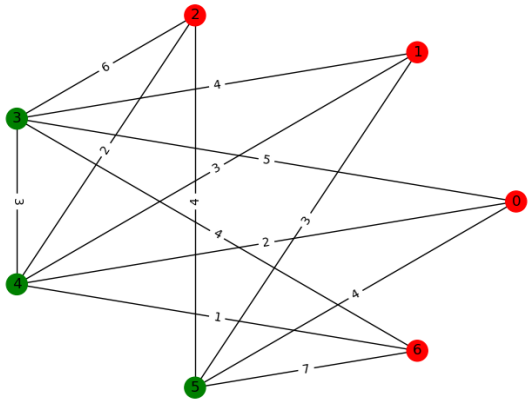
4.1. ábra. Kiindulási hálózat

A 4.1 ábrán látható a bemeneti hálózat, aminek a demand mátrixa már korábban fel lett írva. Az algoritmus ezek után a cl-DAN algoritmust használva, először elkészíti az egófákat 4.2 ábra, majd végül az új hálózat választási sémát, ami a 4.3 ábrán látható.



4.2. ábra. Egófák

A 4.3 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 fokú 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 $\Delta = 12\rho = 12 \cdot \lceil \frac{25}{7} \rceil = 43$.



4.3. ábra. Új hálózat

Egy érdekes tény még megfigyelhető a 4.3 gráfon, mint a konfigurációnál is említve volt, azoknak a csomópontoknak adjuk meg a maximális fokszámát, ami a magas fokszámúak halmazába fog kerülni. Mivel a cl-DAN utolsó lépésében meg van említve, hogy nem csak magas-magas és magas-alacsony kapcsolatok között van él, hanem az alacsony-alacsony között is. Ez annyiban hat ki a végső útválasztási sémára, hogy ha van két telített segítő pontunk, akkor ellenőrzés nélkül be kell húzni az alacsony-alacsony kapcsolatot, ezzel megszegve azt a feltételt, hogy egy csomópontnak csak meghatározott Δ fokszáma lehet. Ha Δ nagyon kicsi, akkor ez a probléma hatványozottan érvényesül, de ha választunk egy megfelelően nagy és reális fokszámot, akkor ez a probléma nagy valószínűséggel nem fog fellépni.

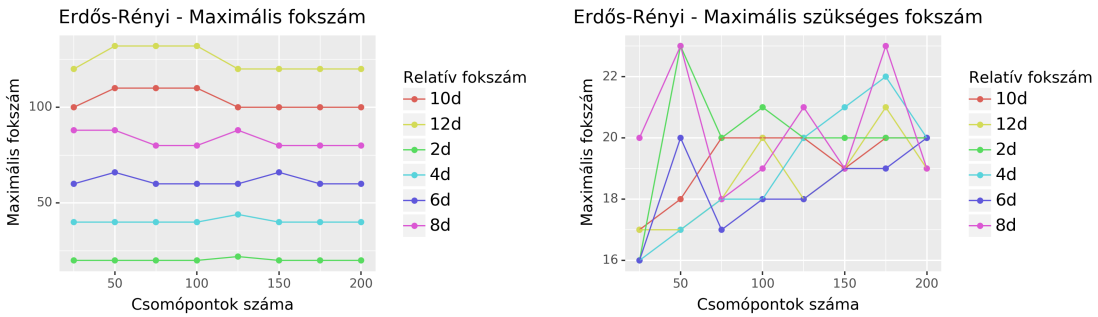
5. fejezet

Tesztelés

5.1. 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 elemeit veheti fel $\Delta \in [10, 16, 24, 48]$. A tesztek segítségével ki lett mutatva, hogy egy ponton túl már annyira nagy lesz a Δ értéke dinamikus esetben, 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 5.1 ábrán látható ez:



5.1. 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 a fent említett

paraméterek összes kombinációjára húsz teszt futott le. Összességében 384.000 teszt készült el, amit meg lehet találni a projekt GitHub oldalán.

5.2. Metrikák

Tesztek különböző metrikák alapján lettek kiértékelve, amik 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

6. fejezet

Teszt eredmények kiértékelése

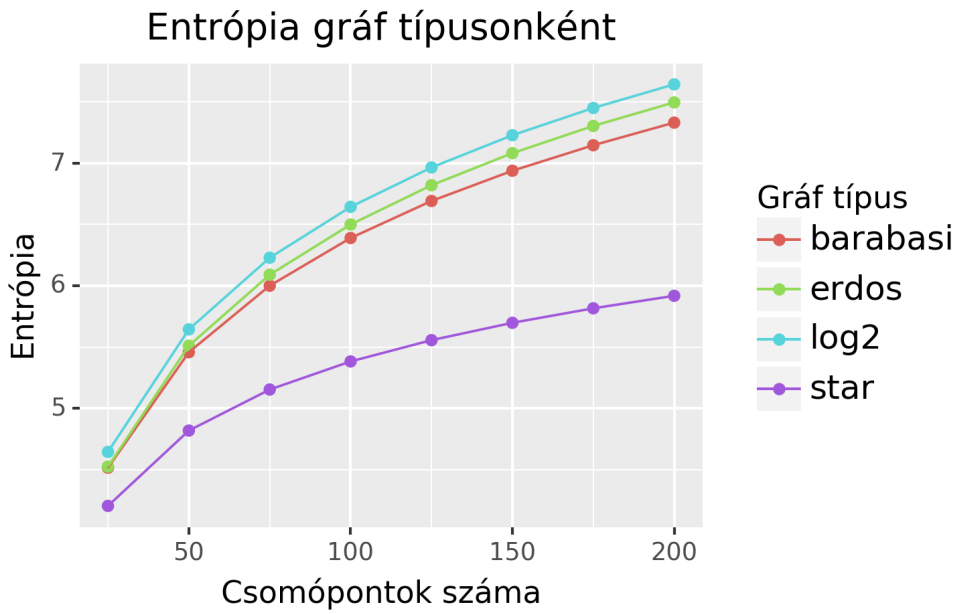
6.1. Entrópia

A különböző gráf típusok különböző entrópiával rendelkeznek. Az entrópia diszkrét valószínűségi változóra felírható a következő képlet [10]:

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

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 entrópia 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 6.1 ábrán.



6.1. ábra. Entrópia

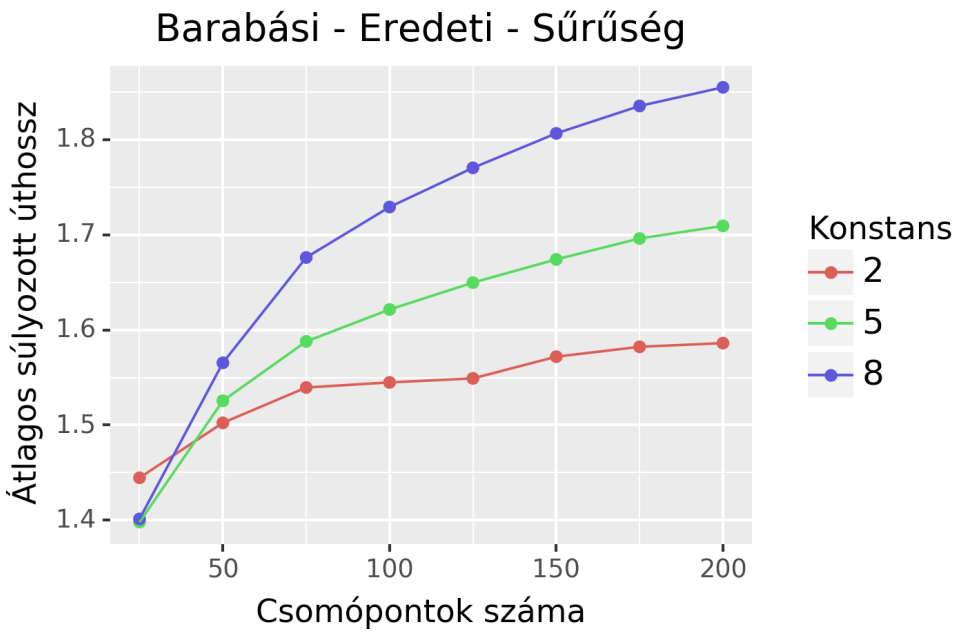
A grafikonon látható, hogy az Erdős-Rényi gráfnak van a legnagyobb entrópiája, itt egy valószínűség változó határozza meg mennyi éle lesz a gráfnak. Ezt követi a Barabási-Albert gráf, ahol tudjuk mennyi élt várunk, annak függvényében mennyi régi csomópontra kell kapcsolódnia az új csomópontnak. Majd végül a legkisebb entrópiát a csillag eredményezte, mivel a csúcsok csak a csillag középpontokhoz csatlakoznak, máshova nem. A $\log_2 n$ pedig az elméleti felső korlát.

6.2. Úthossz

6.2.1. Általános eset

A tesztek során a véletlen gráfok típusától függően a konstans érték határozta meg, hogy mennyire volt kitöltve a demand mátrix. A Barabási-Albert gráf esetén a konstans meghatározta a "preferential attachment" számát, Erdős-Rényi gráf esetén meghatározta valósínűségét annak, hogy két csomópont kapcsolatban áll, és a csillaggráf esetén pedig a csillagok számát adja meg. Ezért először nézzük meg mennyire van kihatással a mátrix kitöltöttsége az eredményekre.

Eredmény a 6.2 ábrán.



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

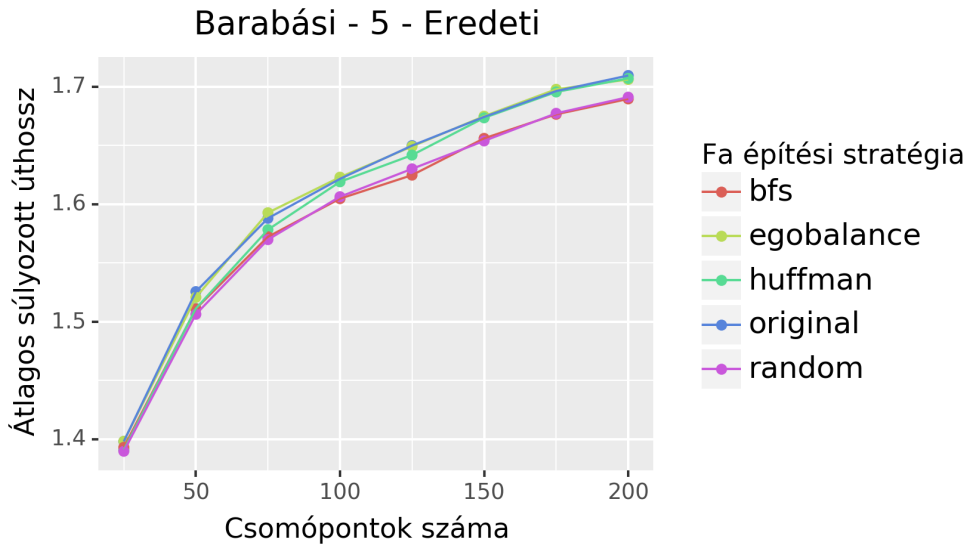
A grafikonon a Barabási-Albert gráf eredményei láthatók, az eredeti mennyiségű fa megépítésével, azaz legalább minden második csúcsponthoz elkészült, és a fák pedig az eredeti algoritmussal készültek el. A hálózatban az összes szereplő él súlya 1, normalizálás után pedig $\frac{1}{|E|}$. Mint látható 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.

6.2.2. 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.

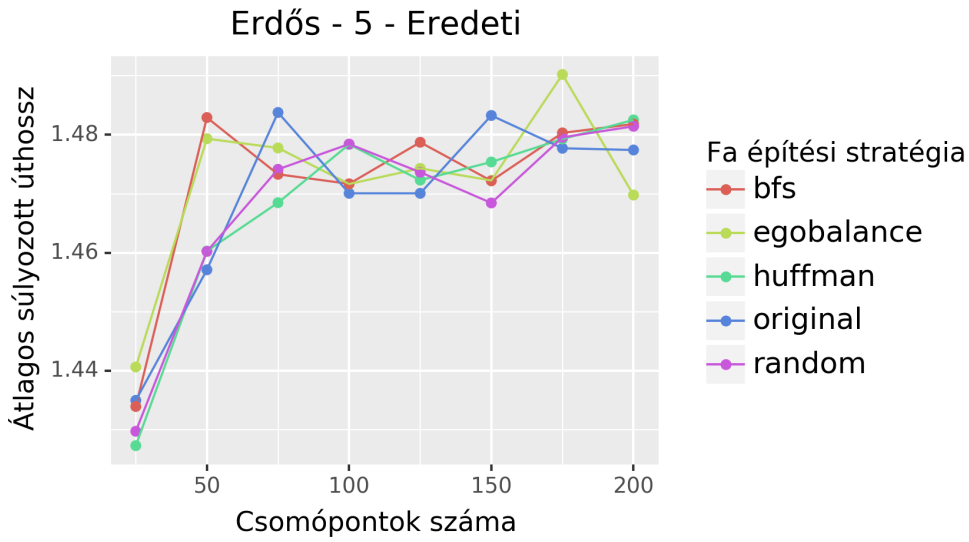
Első véletlen gráf típus a Barabási-Albert gráf, ahol az eredeti számú fát építettük meg. Eredmény a 6.3 ábrán.



6.3. 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 a legkisebb fát, mivel egy teljes fát épít az algoritmus.

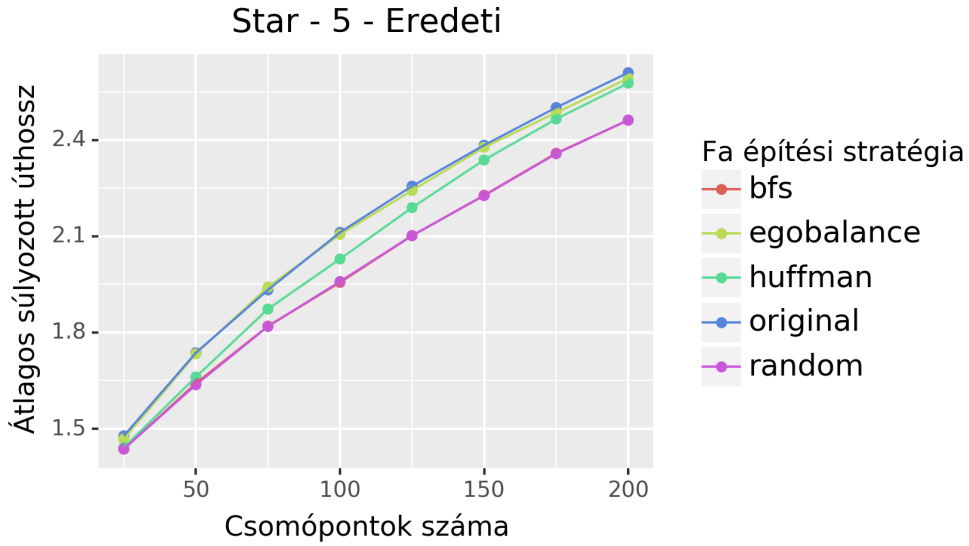
Következő véletlen gráf típus, amit vizsgálunk, az az Erdős-Rényi gráf, itt is az eredeti számú fát építjük meg. Eredmény a 6.4 ábrán.



6.4. ábra. Erdős-Rényi gráf - Úthossz

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

Végül pedig nézzük meg a csillaggráfot az eredeti számú fa mennyiséggel a 6.5 ábrán.



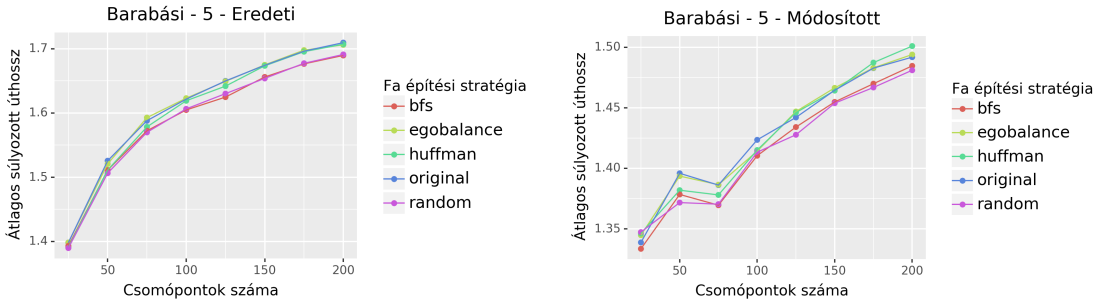
6.5. ábra. Csillaggráf - Úthossz

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, amelyek ténylegesen nagyfokúak, teljesül a magas fokszámú pontokra, hogy a fokszámuk legalább kétszerese az átlag fokszámnak.

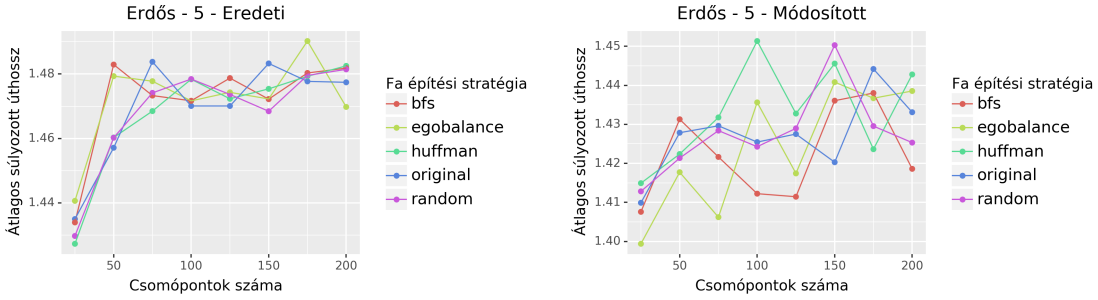
Hasonlítsuk össze a Barabási-Albert gráf eredményeit a 6.6 ábrán.



6.6. 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.

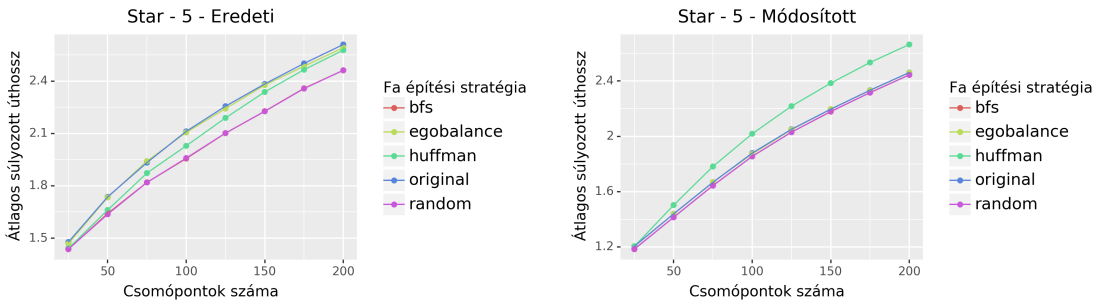
Következő gráfunk az Erdős-Rényi gráf a 6.7 ábrán.



6.7. 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 csillaggráfot a 6.8 ábrán.



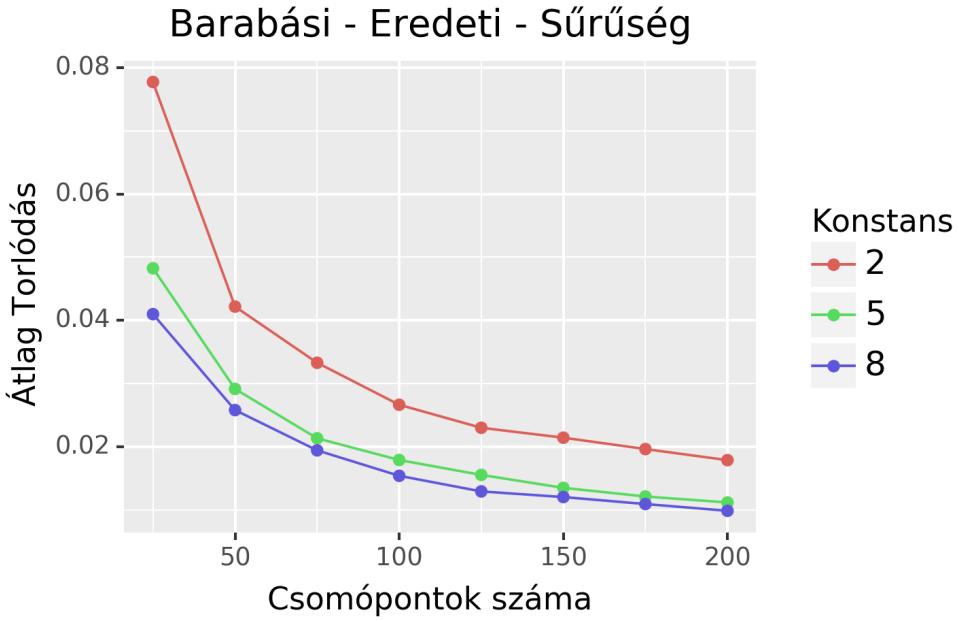
6.8. ábra. Csillaggráf - Megépített fa számosságának összehasonlítása

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 eredményt produkált. A másik négy algoritmus meg javult az úthosszara nézve és megmaradt a relatív pozíciójuk.

6.3. Torlódás

6.3.1. Általános eset

Az úthosszhoz hasonlóan először nézzük meg, hogy az eredeti algoritmus milyen eredményt ad, attól függően mennyire sűrű a gráf. Eredmény a 6.9 ábrán.



6.9. ábra. Torlódás és demand mátrix kitöltöttségének kapcsolata

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

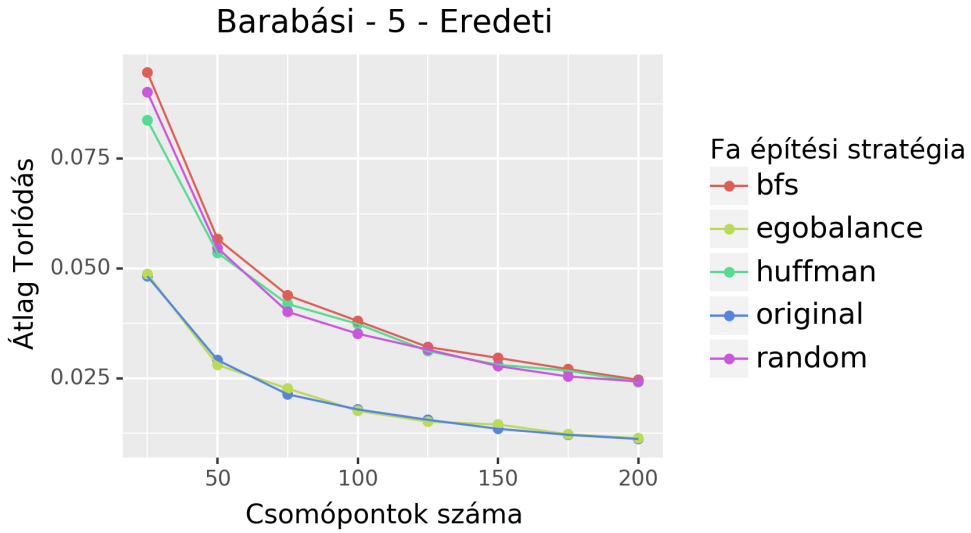
épített fa mennyiséggel, és a fák pedig az eredeti algoritmussal készültek el. A hálózatban szereplő összes él súlya 1. Mint látható a grafikonon, minél ritkább a mátrix, annál nagyobb a torlódás. Egy fontos észrevétel a két metrika között, míg az úthossz átlagosan az egész mátrixra nézve adta meg az 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 torlódásra.

6.3.2. 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.

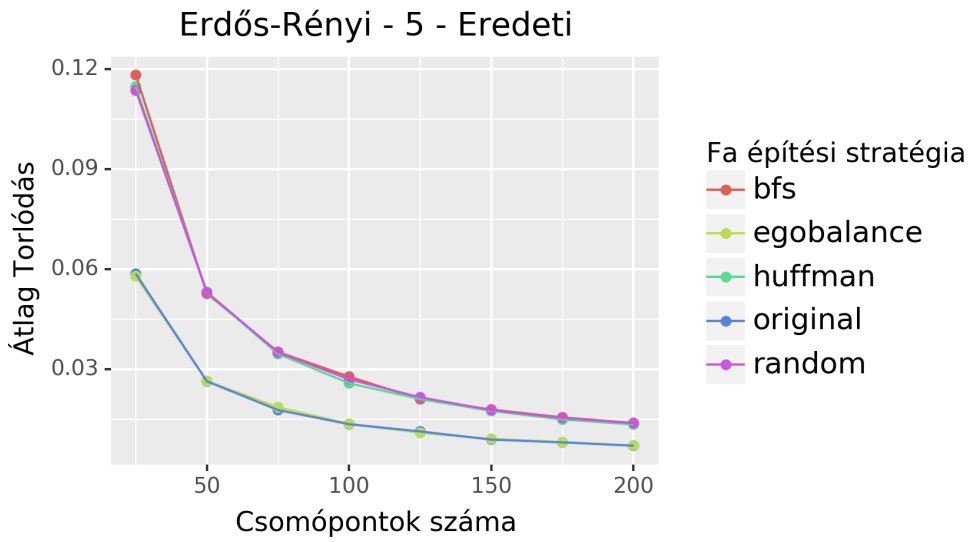
Első véletlen gráf ismét a Barabási-Albert gráf, ahol az eredeti mennyiségű fát építjük meg. Eredmény a 6.10 ábrán.



6.10. 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.

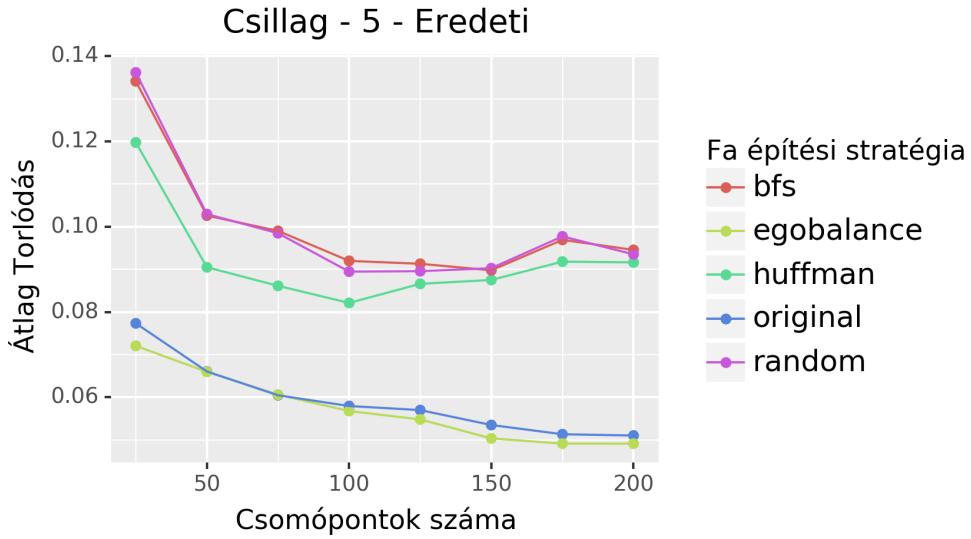
Következő véletlen gráf típus az Erdős-Rényi gráf, itt is az eredeti számú fát építjük meg. Eredmény a 6.11 ábrán.



6.11. 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 csillaggráfot az eredeti fa mennyiséggel a 6.12 ábrán.



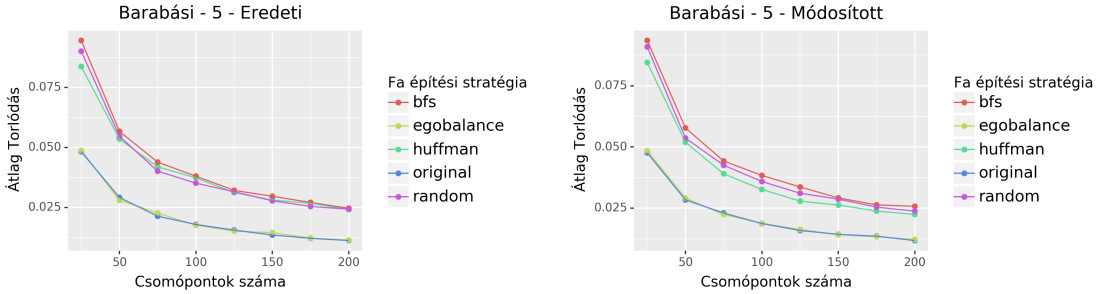
6.12. ábra. Csillaggráf - Torlódás

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

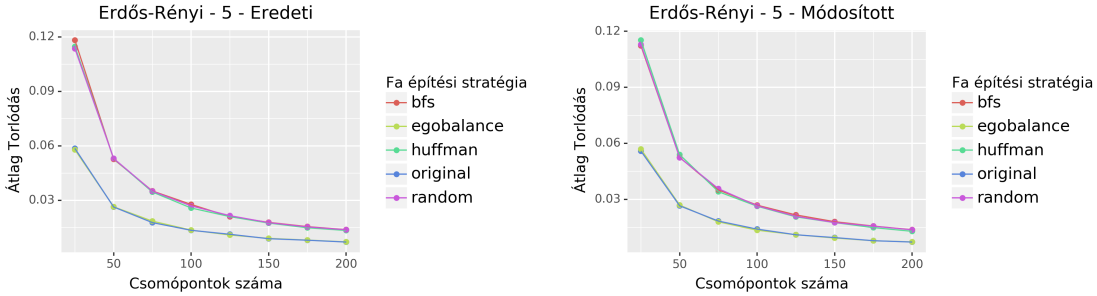
Az előző részben láthattuk milyen eredményeket adnak az eredeti feltétel alapján a különböző algoritmusaink. Most nézzük meg, ha változik a torlódás a megépített fák mennyiségének függvényében. Azokat a fákat építjük meg amelyek ténylegesen nagyfokúak.

Hasonlítsuk össze a Barabási-Albert gráf eredményeit a 6.13 ábrán.



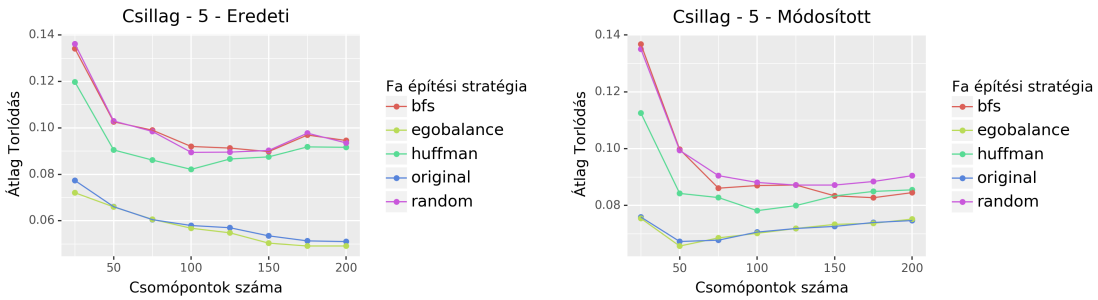
6.13. ábra. Barabási-Albert gráf - Fa számosság összehasonlítás

Számottevően jelentős különbség nem jelentkezik a két eset között.
Következő gráfunk az Erdős-Rényi gráf a 6.14 ábrán.



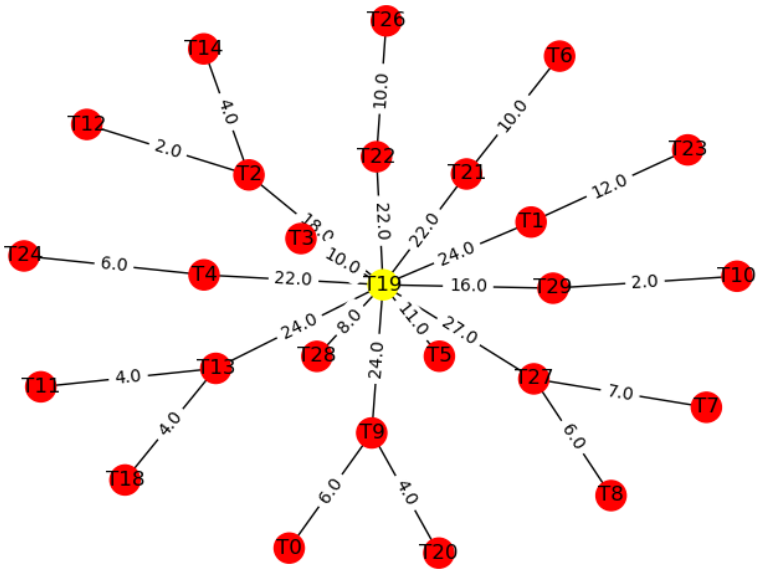
6.14. ábra. Erdős-Rényi gráf - Fa számosság összehasonlítás

Jelentős különbség itt sem figyelhető meg.
Végül nézzük meg a csillaggráfot a 6.15 ábrán.



6.15. ábra. Csillaggráf - 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 6.16 ábra.



6.16. ábra. Csillaggráf - Egófa az eredeti algoritmus alapján

A hálózat ami ezt az Egófát 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 össze, 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.

7. fejezet

Összefoglalás

7.1. Diplomamunka eredménye

A diplomamunka kiinduló pontjául szolgált a Demand-Aware Network Design with Minimal Congestion and Route Lengths [11] cikk. A szerzők arra kerestek megoldást, hogy lehetne csökkenteni a torlást és az úthosszt az adatközpontokban. A felvázolt megoldásuk mellett adtak egy felső korlátot, amivel az algoritmus ad egy közel optimális megoldást. A megvalósított munkában széleskörűen lett tesztelve az algoritmusra adott felső korlát. Ez mellett módosított algoritmusok is hozzá lettek adva a teszteléshez, amik hivatottak voltak az úthosszra nézve jobb eredményt produkálni. Ezek teljesítménye jobb volt úthossz szempontból, ám a torlódás terén nem teljesítettek kiválóan.

7.2. Keretrendszer

A diplomamunkában megvalósított keretrendszer segítségével a szerzők megoldása gyakorlati tesztelés alá lett vetve. Itt több fajta véletlen gráffal lettek szimulálva a lehetséges hálózatok és az azokon folyó adatforgalom. Az Erdős-Rényi modell segítségével modelleztük azt az esetet, mikor a kommunikáció egyenletesen eloszlik a hálózaton és nem centralizálódik több nagyobb szerver köré a forgalom. A Barabási-Albert modell segítségével már nagyobb klasztereket kaptunk, mivel akik később csatlakoztak, azok mindig választottak még m régi csomópontot. Így a legelsők a hálózatban nagyobb eséllyel kaptak több kapcsolatot. Végül pedig az extrém eset, a csillaggráf, mikor több nagyobb csomópont köré összpontosul a forgalom.

Az így megkonstruált gráfokon az eredmények azt mutatják, hogy a cikkben meghatározott "optimális" eset egy határozottan magasabb felső korlát, mint amire valójában szükség van. A megadott $\Delta = 12\rho$ fokszám egy túl nagy felső korlát, mivel olyan gyorsan nő ez a szám, hogy már $\Delta = 4\rho$ esetben is, az algoritmus lefut-

tatása után bőven meghaladja a valóban szükséges fokszámot. Ez azt eredményezi, hogy az összes kapcsolat egy köztes segítő csomópont segítségével majdnem direkt kapcsolódik. Ezért nem tudnak elég magas fák kialakulni, hogy a különböző algoritmusok érdemi különbséget nyújtsanak. Így tesztelés során, egy jól meghatározott korlát segítségével voltak a tesztek lefuttatva. A korlát alapját a piacon jelenleg forgalomban lévő Ethernet switch portok száma adta, ami $\Delta \in [10, 16, 24, 48]$ lett.

7.3. Algoritmusok

A cikkben megfogalmazott cl-Dan algoritmus hagy némi helyet az implementálónak a részletek terén, mivel nem teljesen írja le egy helyzetben a helyes lépést. Ez alapján született két algoritmus, az Eredeti és az EgoBalance, ami a csere lépésben felmerülő szülő nélkül maradt gyerekek újra csatlakoztatását hivatott megoldani. Itt az Eredeti helyben mozgatta át a nehezebb gyereket a szülő pozíciójára, addig az EgoBalance egy természetesebb, az alapul szolgáló adatstruktúrára bízta az újra eloszlást. Eredmény szempontjából a kettő algoritmus nagyon hasonló, és lényegbeli különbség a mérési hiba korlátain belül szerepel.

Annak érdekében, hogy valós hozzávetési alapunk legyen a módszer teljesítményéhez, további másik fa építési módszerek is meg lettek vizsgálva. Ezek alapjául a Huffman fa szolgált és ezekből lett három különböző algoritmus, amelyek rendre a Huffman-, Sorfolyotonos- és Véletlen fák voltak.

A Huffman kódolásban is használt fa lett az első algoritmus, ami az úthosszt hivatott javítani. Ennek eredménye helyenként tényleg jobb lett, mint az Eredeti esetben, de mivel nem csak ez az egy szempontból kell jól teljesíteni, így figyelembe kell venni a torlódást is. Ebből a szempontból már határozottan nem teljesített jól, mert a Huffman fa ágai között akár másfélszeres szorzó különbség is előfordulhat, így leterhelt utak jönnek létre.

Következő algoritmus ami létrejött az a Sorfolytonos fa. Itt egyetlen szempont volt figyelembe véve, építsünk egy teljes fát. Ezzel a módszerrel mindig a legkisebb fák jöttek létre és így ez eredményezte mindig a legrövidebb utakat. Torlódás szempontjából ugyanolyan rosszul teljesített mint a Huffman fa, mivel itt sem volt figyelembe véve ez a szempont. Mivel ez volt a legjobb úthosszra adott algoritmus, lehet érdemes lenne további kutatást folytatni annak érdekében, hogy hogyan lehet a torlódást javítani egy ilyen hálózatban.

Végül a Random fa, ami nem különbözik különösebben a Sorfolyotnostól. Ennek az eredménye sem lett jobb, mint amire épül, ám lehetőség van arra, ha épp úgy helyezkednek el az elemek, hogy az torlódás szempontjából is kedvező, akkor van némi esély arra, hogy jobb eredményt ad.

Ezekből az eredményekből az látszik, hogy a cikkben szereplő algoritmus megfe-

lelő a céljára, ám ez nem zárja ki annak a lehetőségét, hogy egy másfajta javításibeli megközelítést is kivizsgáljunk.

7.4. Magas fokszámú csomópontok

A második javítási mód előtérbe helyezi azt a szempontot, hogy mérlegeljük a hálózatban szereplő ténylegesen leterhelt pontokat. Ezzel célzottabban tudjuk a hálózat magas fokszámú csomópontjait azonosítani és elejét vehetjük az olyan esteknek, ahol nincs szükség átépítésre. Ezzel csökkenthetjük a számítás igényét az algoritmusnak és egyúttal még rövidebb utakat is érünk el a legtöbb esetben. Mivel ez a megközelítés csak finomítja a cl-Dan első lépését, ami alapján osztályozzuk a pontokat, így legrosszabb esetben is visszaértünk a kiindulási algoritmushoz és soha nem lesz rosszabb attól. A mérések azt mutatják, hogy az úthossz határozottan csökken, de a torlódásra az esetek túlnyomó részében nincs kihatással, ám vannak elfajult esetek, ahol rosszabbat eredményez.

8. fejezet

Köszönetnyilvánítás

Szeretnék köszönetet nyilvánítani témavezetőmnek Lukovszki Tamás, PhD egyetemi docensnek, amiért felajánlotta és vezetett a téma kidolgozásában és megvalósításában. Továbbá még szeretném megköszönni Veress Orsolya támogatását is, a munka korrektúrájában nyújtott segítségéért.

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

[5] Python - Python.org.

[6] Wikipédia - Barabási–Albert-modell.

[7] Wikipédia - Csillaggráf.

[8] Wikipédia - Erdős–Rényi modell.

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

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

[11] 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.

[12] P. Erdös and A. Rényi. On random graphs i. *Publicationes Mathematicae Debrecen*, 6:290, 1959.

[13] 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.

[14] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17:416–429, 1969.

[15] 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.

[16] E. L. Schreiber, R. E. Korf, and M. D. Moffitt. Optimal multi-way number partitioning. *J. ACM*, 65(4):24:1–24:61, July 2018.