

# Forgalom igény tudatos hálózat tervezés minimális torlódással és úthosszal

Tudáskezelő rendszerek II. labor összefoglaló

Szecsődi Imre

2019

# Tartalomjegyzék

<b>1. Bevezetés</b>	<b>3</b>
1.1. Motiváció . . . . .	3
1.1.1. Hálózat tervezési stratégiák . . . . .	4
1.1.2. Adattárházak hálózati felépítése . . . . .	4
1.1.3. Újrakonfigurálás megvalósítása . . . . .	5
1.2. Labor célja . . . . .	5
1.3. Laborban megvalósított munka . . . . .	5
<b>2. Modell</b>	<b>6</b>
2.1. Forgalom igény tudatos hálózat tervezés probléma . . . . .	6
2.2. Formális felírás . . . . .	6
2.2.1. Torlódás . . . . .	7
2.2.2. Úthossz . . . . .	7
2.2.3. Skálázhatóság . . . . .	7
2.2.4. Optimális torlódás . . . . .	7
2.2.5. Optimális úthossz . . . . .	7
2.3. cl-DAN hálózat tervezése . . . . .	7
2.4. EgoTree . . . . .	8
2.5. $EgoTree(s, \bar{p}, \Delta)$ algoritmus . . . . .	8
2.5.1. Algoritmus elemzése . . . . .	9
2.5.2. Longest Processing Time (LPT) . . . . .	9
2.6. cl-DAN algoritmus . . . . .	9
<b>3. Megvalósítás</b>	<b>11</b>
3.1. Keretrendszer . . . . .	11
3.2. Adatszerkezetek . . . . .	11
3.3. Modell . . . . .	12
3.4. Kimenet . . . . .	14
<b>4. Teszt eredmények</b>	<b>16</b>
4.1. Tesztelés menete . . . . .	16

4.2.	Átlag súlyozott úthossz . . . . .	16
4.3.	Átlag torlódás . . . . .	16
4.4.	Fokszám . . . . .	18
<b>5.</b>	<b>Összefoglalás</b>	<b>19</b>
5.1.	Cikk eredménye . . . . .	19
5.2.	Megjegyzések . . . . .	19
<b>6.</b>	<b>Irodalomjegyzék</b>	<b>20</b>

# 1. fejezet

## Bevezetés

A labor munka a Demand-Aware Network Design with Minimal Congestion and Route Lengths [4] cikk alapján készült.

### 1.1. Motiváció

- A technika előrehaladásával egyre nagyobb lett a feldolgozandó adatok mennyisége
- Adattárházakban a szerverek közötti kommunikáció is ezáltal megnövekedett
- A jelenlegi hálózatok a legrosszabb esetre vannak tervezve, azaz, hogy majdnem teljes sávszélességű, kétirányú kapcsolat álljon fent bármelyik két szerver között
- A valós kommunikáció nem ezt a sémát követi, hanem túlnyomó részt megadott párok között történik a legtöbb kommunikáció

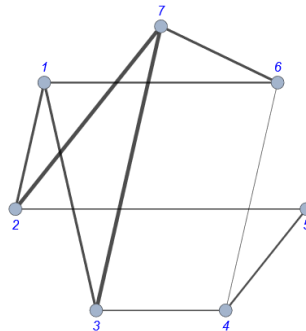
Microsoft Research ProjectToR [5].

- Nézzünk meg pár valós példát, Microsoft adattárházában 250 ezer szervert 5 production klaszterben elosztva

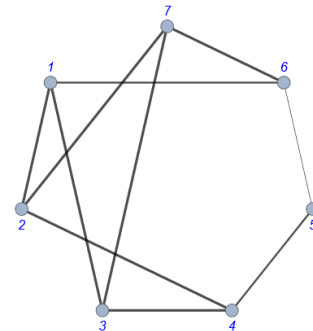
### 1.1.1. Hálózat tervezési stratégiák

	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

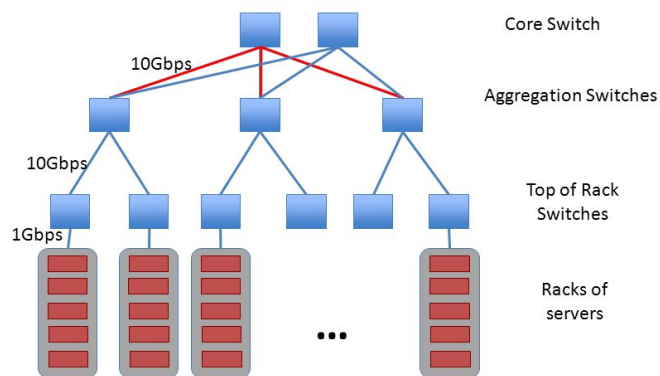


(c) Optimal route congestion

- A technika fejlődésével elérhetővé váltak eszközök arra, hogy egy adott hálózatot újra konfiguráljunk, attól függően milyen terhelés éri
  - pl, korábbi kommunikációs minták alapján
- Két fő optimalizációs lehetőség van, legyen rövid az út (a) vagy legyen minimális a torlódás (b)
- A cikk bemutat egy módszert arra, hogy lehet mindkettőre majdnem optimális megoldást adni egyszerre (c)

### 1.1.2. Adattárházak hálózati felépítése

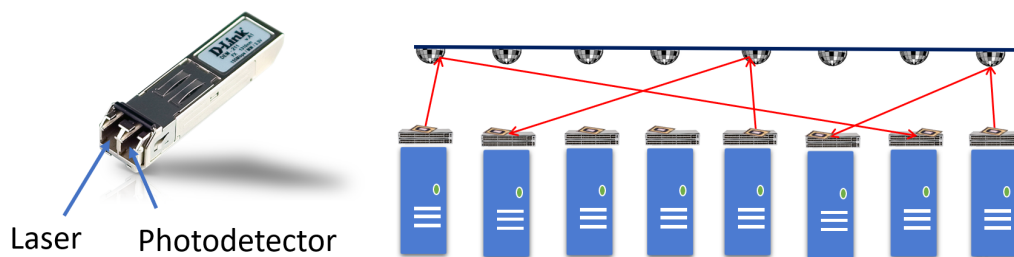
#### Traditional Data Center Topology



- Core switch
- Aggregation Switches
- Top of Rack Switches
- In-Rack Switches

### 1.1.3. Újrakonfigurálás megvalósítása

- Átlag hálózatok statikusan vannak konfigurálva, nem sok lehetőséget adva annak, hogy változtassunk
  - pl. Ethernet switchek
- Optikai switchek már újra tudják konfigurálni magukat, de ezek "lassúak"
- Microsoft Research - ProjecToR[5], lézer segítségével kiváltani az optikai switcheket, 1.1 ábrán látható a maga az eszköz
  - $12\ \mu s$  váltás idő ( 2500x gyorsabb mint egy optikai hálózati switch)



1.1. ábra. ProjecToR

## 1.2. Labor célja

A labor célja a cikkben[4] bemutatott algoritmus implementálása, és annak alkalmazása különböző véletlenszerűen generált gráfokra. A kapott eredményeket össze lehet hasonlítani 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 [3] nyelven íródott. Egy véletlen gráfok generálására egy külső csomag lett használva [2]

## 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

### 2.2. Formális felírás

- Adott  $N$  darab csúcspont  $V = \{1, \dots, N\}$ , 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  $\Gamma_{uv}$  utak halmaza, ahol  $(u, v)$  párok különböző utakat jelölnek
- $\Gamma_{uv}$  egy útsorozat, ami összeköti az  $u$  pontot  $v$  ponttal

### 2.2.1. Torlódás

**1. Definíció.** 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. Definíció.** 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  $\Delta$  konstans fokszámot, ami a maximális csatlakozások számát fogja meghatározni egy adott csomóponthoz
- $N_\Delta$  jelölje az összes  $\Delta$  fokszámú gráfot, és elvárjuk, 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 számításakor

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

### 2.2.5. Optimális úthossz

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

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

## 2.3. cl-DAN hálózat tervezése

**3. Definíció.** Adott egy  $D$  demand mátrix, és egy  $\Delta$  maximális fokszám, az  $(\alpha, \beta)$ -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
- $\Delta$  fokszám

- Ez közel optimális megoldást ad torlódásra és úthosszra

**1. Tétel.** *Adott egy  $\bar{p}$  frekvencia eloszlás az  $s$  forrás ponthoz, és adott egy  $\Delta$  fokszám, ekkor az  $EgoTree(s, \bar{p}, \Delta)$  egy  $(\alpha, \beta)$ -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,  $\Delta$  fokszámmal, üres fa
2. Rendezzük sorba  $\bar{p} = \{p_1, p_2, \dots, p_k\}$  valószínűségeket csökkenő sorrendben
3. Kezdjük rárakni a fára a csomópontokat, a gyökér elemre legfeljebb  $\Delta$  levél kerülhet
4. Mikor elértük a  $\Delta$  levelet, a következő csomópontokat mindig a legkisebb összeített 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  $\Delta$  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  $\omega_L$  a maximum idő, mielőtt egy processzor befejezi az összes munkát a mohó LPT algoritmus szerint, és  $\omega_0$  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  $\Delta$  fokú fához képest, ami kiszolgál  $\bar{p}$  frekvencia eloszlást egy adott  $s$  forrás csomóponttra

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

## 2.6. cl-DAN algoritmus

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

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

Konstans  $\rho$  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  $2\rho$
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. Meghatározzunk 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. Networkx csomag továbbá biztosít számunkra egy megjelenítési lehetőséget, amit a Jupyter notebookban tudunk legjobban kihasználni.

### 3.2. Adatszerkezetek

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

- **Vertex** - az általános gráf csúcspont
- **Node** - az *Egófák* készítésekor használt csomópontok amik tartalmazzák a valószínűségét annak, hogy a forrás csomópont mekkora valószínűséggel fog kommunikálni a másik *Node* csomóponttal
- **Edge** - az gráf csomópontjait reprezentáló él, ami *Vertexet* vár paraméterként, és tárolja a valószínűséget, hasonlóan mint a *Node*
- **Tree** - ami adja az alapját majd a útvonal tervezési sémának. A fának két fajtája lehet:
  - **BinTree** - a kettő fokú fa
  - **EgoTree** - a  $\Delta$  fokú fa, ahol a gyökérnek legfeljebb  $\Delta$  levele lehet, és a levelek pedig *BinTree* típusúak.

### 3.3. Modell

A **Network** osztály valósítja meg az algoritmust, bemenete egy konfiguráció, kimenete egy útválasztási séma. Ez mellett sok metaadatot is kiszámol a program amik között szerepel az átlag súlyozott úthossz és a torlódás. A konfigurációban lehetőségünk van dinamikus delta fokszámot megadni, ezért metaadatok között szerepel a fokszaám, amit az algoritmus használt számoláskor. Az új hálózat létrehozásakor a csere lépéssorozat után, megváltozik a demand mátrix, ezért fontos, hogy a  $\Delta$  fokszaómot ne haladjuk meg. A metaadatok ezért tartalmazzák a delta fokszaómot és a valós fokszaómot, ami az algoritmus végeredménye lett. Ebből az adatból lehet következtetést levonni, hogy valóban megfelelő-e felső korlátja amit a szerzők adtak és lehet-e jobb felső korlátot adni.

A bemeneti konfiguráció egy *JSON* fájl, amit tartalmazhat több konfigurációt egyszerre. Többféle módon lehet megadni konfigurációt attól függően milyen gráfot akarunk használni. Lehetőség van kézíleg 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 fokszáma ad megkötést. Értékek amit felvehet:
  - *null* - Alap beállítás szerint a cikkben ajánlott  $12\rho$  fokszámot fogja használni
  - *12* - A megadott szám lesz a foksám, 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  $\rho$  szorzója, példa most  $6\rho$

Konfiguráció függő mezők:

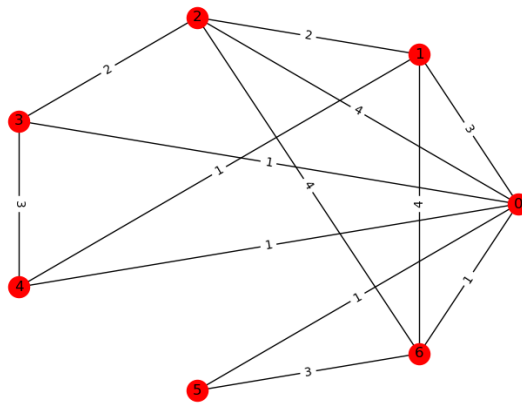
- **constant** - *Erdő-Rényi* gráf esetén a modell vár egy  $p$  valószínűséget ami annak a valószínűsége, hogy egy él be legyen húzva az új pontba, és ez független minden a korábban behúzott élektől. A ritka mátrix eléréséhez a következő formulát használja a generáló algoritmus:

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

- **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!

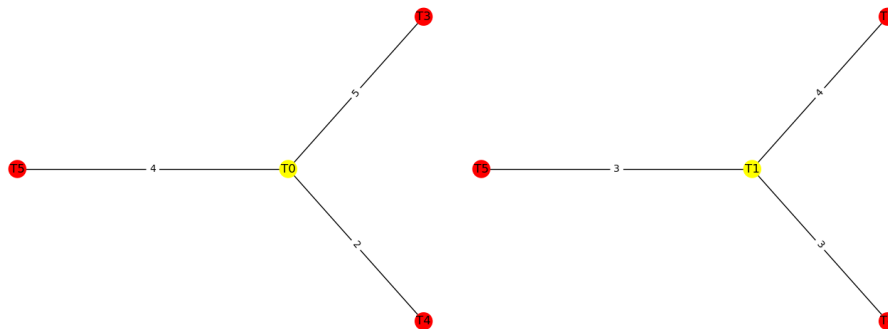
### 3.4. Kimenet

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.



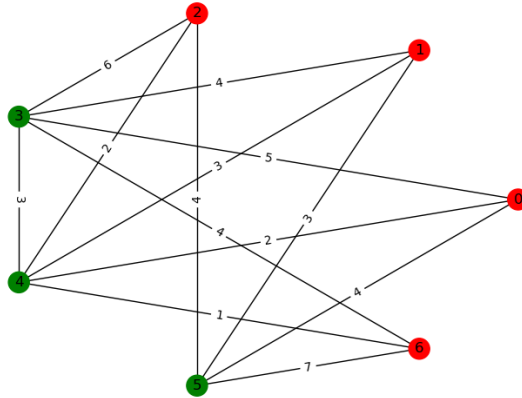
3.1. ábra. Kiindulási hálózat

A 3.1 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 3.2 ábra, majd végül az új hálózat választási sémát ami a 3.3 ábrán látható.



3.2. ábra. Egófák

A 3.3 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 fokszámú csúcs csak egy alacsony fokszámú segítő csúcson keresztül tud kommunikálni. Fokszámok szempontjából a csúcsok rendben vannak, mivel nem haladják meg  $\Delta$  fokszámot. A gráf esetén a delta  $\Delta = 12\rho = 12 \cdot \lceil \frac{25}{7} \rceil = 43$ .



3.3. ábra. Új hálózat



## 4. fejezet

# Teszt eredmények

### 4.1. Tesztelés menete

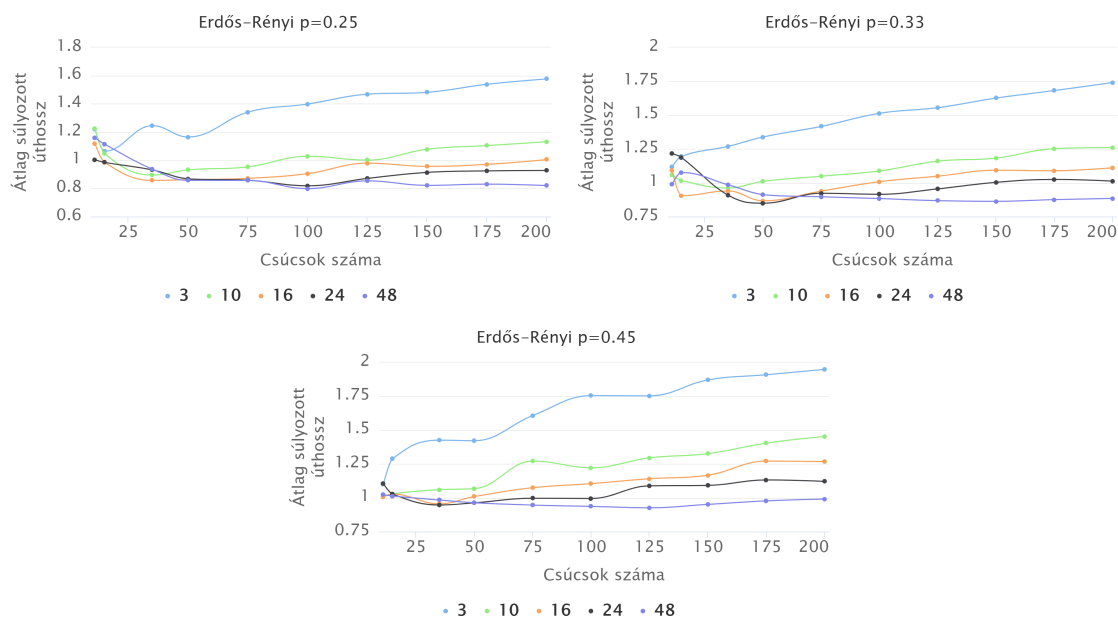
A tesztelés során Erdős-Rényi gráfok voltak használva az egyszerű skálázhatóságból kifolyólag. Három fő csoportba lehet a méréseket osztani a valószínűségi érték alapján,  $p = 0.25$ ,  $p = 0.33$  és  $p = 0.45$ . A skálázhatóság tesztelésére különböző számú csúcspont volt használva, ezek rendre: 11, 15, 35, 50, 75, 100, 125, 150, 175 és 200. A skálázhatósághoz még hozzátartozik a maximum fokszám, hogy mennyi gráf csúcs tud egyszerre kommunikálni, ezért az is meg van adva. A valóságban van egy fizikai határ, hogy mennyi kliens tud csatlakozni egyszerre, ezért az értékek így lettek megválasztva, 3, 10, 16, 24 és 48. A fenti paraméterek összes kombinációjára öt teszt lett futtatva, majd azok átlagolva lettek. Az adatok egy CSV fájlban lettek összegyűjtve, majd RapidMiner segítségével lett kiértékelve.

### 4.2. Átlag súlyozott úthossz

Az 4.1 ábrán látható az átlag súlyozott úthossz csúcspontok függvényében, és a delta fokszámra lebontva. Az átlag súlyozott úthossz mind három esetben hasonlóan néz ki. A konstans fokszám miatt egyértelműen látszik, hogy az értékek növekednek, kicsi delta esetén, pl. 3, ez nagyon jól megfigyelhető. Az eredmények hasonlóak mind három esetben, a kitöltöttség növekedésével az értékek is növekednek, de a grafikon iránya nem változik.

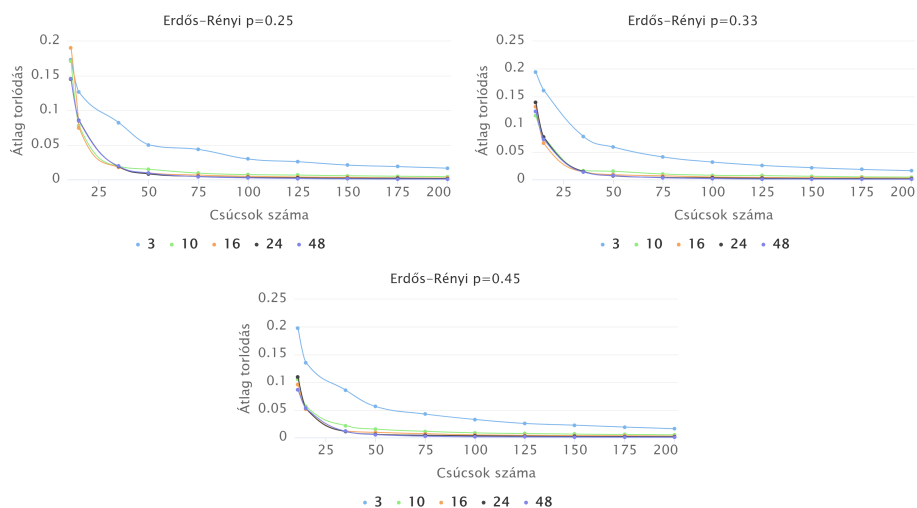
### 4.3. Átlag torlódás

A 4.2 ábrán látható az átlag torlódás. Hasonlóan mint az úthossznál a fő tényezők a csúcsok száma és az, hogy hogyan választjuk meg a fokszámot. Ám ha megnézzük a két diagram irányát, két különböző dolgot mutatnak. A súlyozott úthossz növekedik



4.1. ábra. Átlag súlyozott úthossz

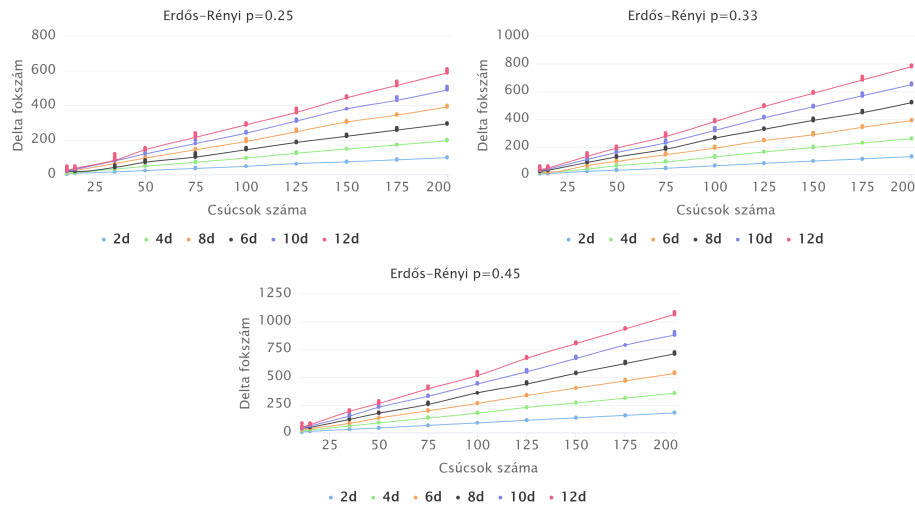
ahogy hozzáveszünk további pontokat a hálózathoz, addig a torlódás mértéke közel exponenciálisan csökken. Ennek egyszerű oka van, mert minél több csomópont közül tudunk választani, a terhelést egyre több pont között lehet elosztani és így a torlódás egészében csökken a hálózatban.



4.2. ábra. Átlag torlódás

## 4.4. Fokszám

A fentiekben már volt említve, hogy adott számú csomópontot használunk. A cikkben szereplő felső korlátok az átlag fokszám függvényében van megadva, ami azt eredményezi, minden hálózatnak más  $\Delta$  fokszám ad közel optimális megoldást. Ez elméletben teljesül is, de ha megnézzük a 4.3 ábrát, rögtön észrevehető egy érdekes jelenség. A delta fokszám majdnem minden esetben nagyobb mint a csúcsok száma ami a hálózatban van. Ez azt eredményezi, hogy minden csomópont közel két lépésből elérhető, és csak akkor van szükség az algoritmus újra rendező részére, 5. lépés, ha egy csomóponttal kommunikál több mint a fele pont a hálózatban. Erre egy példa a 200 pontból álló hálózat, ami a képen látható, hogy már ha a  $4\Delta$  a fokszámot annyi mint a pontok száma a hálózatban.



4.3. ábra. Delta fokszám

## 5. fejezet

# Összefoglalás

### 5.1. Cikk eredménye

Az technológia fejlődésével lehetőség nyílt arra, hogy a hálózatokat újrakonfiguráljuk futás időben. A változó topológiának köszönhetően hatékonyabb lesz az adattárházak működése. Az forgalom igény tudatos hálózatok tervezése körül jelenleg intenzív kutatás folyik ebből kifolyólag. A feldogozott cikk és a benne található algoritmus ad egy megoldást arra, hogy lehet ilyent tervezni. Az algoritmus lényege, hogy a túlterhelt csomópontok között irányítsuk át forgalmat egy segéd csomóponton. Ennek a megvalósítása az egófákkal van megadva, ahol lekorlátozzuk a maximális csatlakozások számát a jobban terhelt szerverekhez. A szerzők által meghatároztak egy felső korlátot az átlag súlyozott úthosszra és torlódásra megadott fokszám mellett ami valóban megfelelő.

### 5.2. Megjegyzések

A teszt eredményekből látszik, hogy a  $12\Delta$  fokszám egy nagyon magas korlát, mivel bőven túlmutat a rendelkezésre álló csúcsok számán. Feltételezhető, hogy ettől lehet jobb felső becslést is adni. Egy érdekes megfigyelés az algoritmus esetén, ha egy olyan fokszámot adunk meg, ami nem mutat túl a létező csúcspontok számán, hanem ellenkezőleg, elég szigorúra van véve. Ilyenkor csak a magas fokszámú pontokra van garantálva, hogy a megadott fokszámú ponthoz csatlakoznak és ez nem fog teljesülni az alacsony fokszámúakra. Egy ilyen eset legegyszerűbben akkor fordul elő, ha már telített a fokszáma a segítő pontnak magas fokú pontokkal. Mivel nem csak magas fokú csomópont fog kommunikálni egy segítővel, ezért még azokat is be kell húzni, mikor két segítő fog kommunikál, és ilyenkor nincs figyelve arra, hogy mi a jelenlegi fokszám.

## 6. fejezet

# Irodalomjegyzék

- [1] Jupyter - <https://jupyter.org/>.
- [2] NetworkX - <http://networkx.github.io/>.
- [3] Python - [Python.org](http://python.org).
- [4] C. Avin, K. Mondal, and S. Schmid. Demand-Aware Network Design with Minimal Congestion and Route Lengths. page 9.
- [5] 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.