TDK dolgozat

Útvonalválasztás rugalmas spektrumú optikai hálózatokban

Budapesti Műszaki és Gazdaságtudományi Egyetem
Távközlési és Médiainformatika Tanszék

Jakab Csaba

Konzulens: Dr. Cinkler Tibor

Bevezető

Az OFDM (orthogonal frequency divison multiplexing) egy modulációs technika, mely több alacsonyabb sebességű vivőfrekvenciát alkalmaz,így lehetőséget ad a spektrumkihasználtság javítására, rugalmas spektrum lefoglalásra. Emiatt az optikai hálózatok terén kutatók figyelme az utóbbi évben erre összpontosult, az OFDM-et ígéretes technikának tartják a jövő nagysebességű optikai átviteleihez.

Az egyes éleken a rugalmas spektrum lefoglalása eddig nem volt szempont az útvonalválasztásnál, ezért tudomásunk szerint megfelelő módszerek erre nem is születtek. Ezért javasoltuk spektrumfüggő legrövidebb útválasztás (Spectrum-aware Shortest Path Routing) módszerünket. Az optikai OFDM hálózatot egy gráffal modellezzük mely élei tartalmazzák a spektrumot és két tetszőleges csúcs között kell az adott igénynek megfelelően az optimális útvonalat keresni(spektrumot lefoglalni) az O-OFDM előnyeit kihasználva. Az adott algoritmust c++ környezetben a LEMON függvénykönyvtár használatával valósítottuk meg és szimulációs vizsgálattal támasztottuk alá hatékonyságát.

Az elmúlt időszakban az optikai gerinchálózatok forgalma évről évre jelentősen növekszik és feltehetőleg a jövőben is exponenciálisan növekedni fog a HD-televíziózás és valós idejű videó kommunikáció széleskörű terjedése következtében

E növekvő forgalmi igényeket kell elvezetni az optikai gerinchálózatban mely jelenleg WDM(Wavelenght Division Multiplexing) azaz hullámhossz osztásos multiplexálást használ. Célunk a sávszélesség maximalizálása.

Technológiai háttér

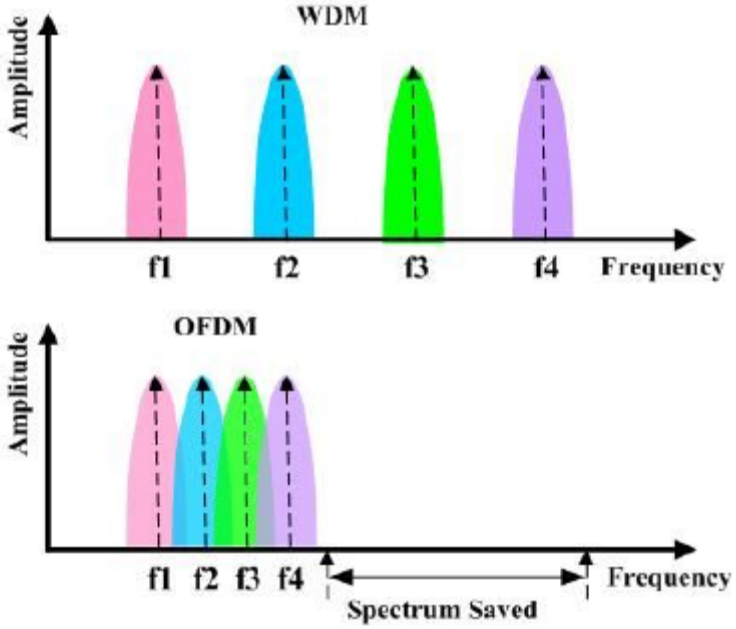
Az Optical Orthogonal Frequency-Division Multiplexing O-OFDM egy modulációs technológia. Az OFDM moduláció alkalmazása, a vezeték nélküli és vezetékes hálózatokban hatékonynak bizonyult, ezért most az optikai hálózatokban került figyelembe. Az OFDM előnyeivel egy új, rugalmas, a jövőbeni elvárásoknak megfelelő optikai hálózati architektúra építhető ki mely lehetővé teszi az optikai jelek kezelését változó adatsebességgel és sávszélességgel.

OFDM Előnyei:

-az OFDM egy speciális sok-vivőfrekvenciás modulációs eljárás mely nagy sebességű adatfolyamot tud továbbítani, azt több alacsonyabb sebességű ortogonális csatornára bontva.

-WDM-el (Wavelength Division-Multiplexing) összehasonlítva itt nincs szükség fix csatornakiosztásra.

Az 1. ábrán látható, hogy az OFDM-nek az ortogonalitás miatt jobb a spektrumkihasználtsága.



1. ábra

Egy WDM-et és egy OFDM-et használó szál spektruma

forrás [1]

A WDM hálózatokban kötött frekvencia kiosztást használnak, tehát az egyes csatornák egymástól fix távolságra vannak. Ha egy csatornának nem maximális a kihasználtsága akkor nagy kihasználatlan spektrumsávot kapunk amit a kötöttség miatt a szomszédos csatornák sem tudnak használni.[1]

Az OFDM alapú hálózatokban az egyes optikai utak között kell hagyni egy Filter Guard Band-ot(védősáv).
A 2.ábra jól szemlélteti az OFDM rugalmas spektrumkiosztását. Itt egy optikai szál spektrumát láthatjuk.[2]

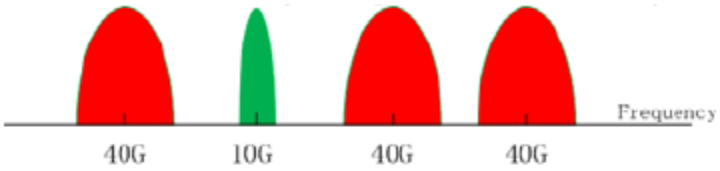


Fig. 1. Spectrum of WDM networks.

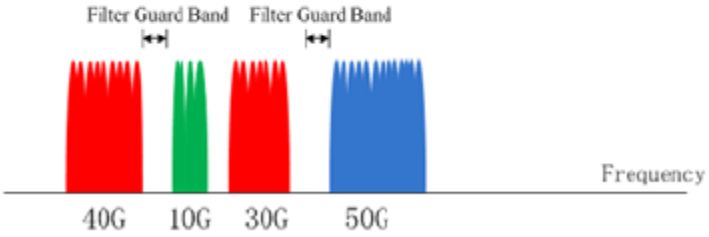


Fig. 2. Spectrum of Spectrum-Elastic Optical Path Networks.

2. ábra
WDM és OFDM csatornák
forrás[3]

A modellben felhasznált OFDM jellemzők:

1. Egy kérés által igényelt sávszélesség bárhol lefoglalható a spektrumban.
   Az OFDM fő előnye a rugalmas spektrum, nincs a WDM-hez hasonló kötött sepktrumkiosztás.

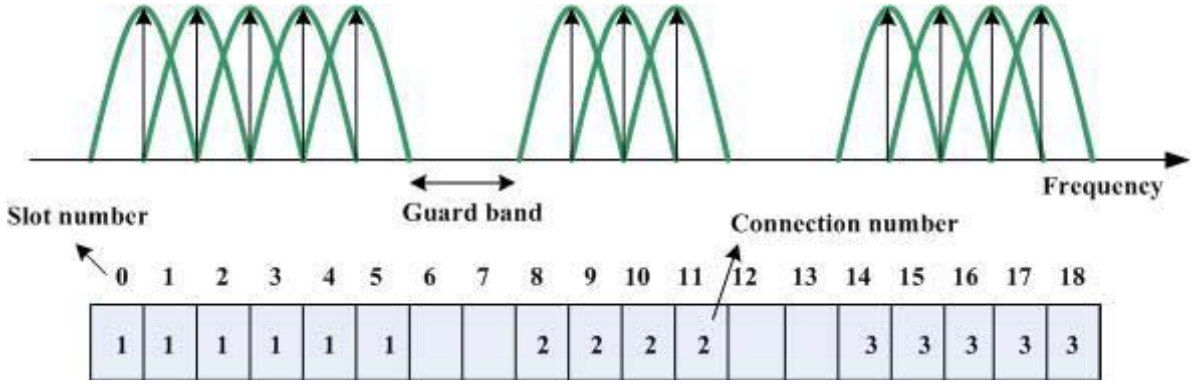
2. Különböző utak között Guard Band[3]
   Egy optikai szál spektrumában, két egymással szomszédos optikai út(a használt vivőfrekvenciák a spektrumban egymás mellett vannak) között egy csatorna üres.

3. Sok alacsony sebességű vivőfrekvencia[3]
   A vivőfrekvenciák összefogásával kapunk egy szélesebb spektrumszálat ami egy optikai úthoz tartozik

4. Két pont közötti optikai út sávszélessége tetszőlegesen széles lehet[1]
   A forgalomtól függő szélességű spektrumszeletet foglalunk le, dinamikusan változik a spektrum

Probléma ismertetése

Egy optikai út mentén adott időintervallumban több kérés is érkezik, ugyanakkor egyes forgalmak csökkenek is, tehát dinamikusan változik a használni kívánt sávszélesség.



3. ábra
a modellben használt él spektrumának szemléltetése
forrás[1]

Több probléma is felmerül az útválasztással kapcsolatban:

1.) Egy kérést két csomópont között kell elvezetni, amire több lehetséges útvonal is van, a kisebb energiafelhasználás és spektrumtakarékosság szempontjából is a legrövidebb út mentén célszerű elvezetni a forgalmat.

2.) Spektrum töredezettsége: Ha találtunk k számú lehetséges útvonalat, hogyan tudjuk elvezetni a forgalmat? Mely vivőfrekvenciákat foglaljuk le. Az algoritmus úgy allokálja a spektrumsávot, hogy kicsi legyen a szemcsésség, ne legyenek nagy hézagok a spektrumban.

3.) A blokkolást kerüljük el, azaz ha a legrövidebb út mentén nincs szabad sáv akkor máshol is legyen elvezethető a forgalom.

4.) Ha egy út mentén csökken a forgalom akkor a felszabaduló frekvenciákat egy másik út mentén is ki lehessen használni.

Rugalmas spektrumú hálózatokban ilyenkor a kérésnek megfelelő sávszélességet kell biztosítani. A hálózatban a teljes optikai út mentén kell lefoglalni a megfelelő számú vivőfrekvenciát.

A modell

Az optikai gerinchálózatot egy irányítatlan gráffal modelleztem. A gráf csúcsai a BV-WXC (Bandwith Variable Wavelength Cross Connect)[1]-k és átjárók az elektromos rétegbe. A kérések mindig két csúcs között érkeznek.

A gráf élei adják az optikai központok közötti optikai szálakat. A gráf éleihez rendeltem az optikai szálak spektrumát. A spektrum állapotát egy bináris 0-1 sorozat jelképezi, ahol az egyes számok a megfelelő OFDM vivőfrekveniát jelentik. Logikai igaz értékkel jelöltem azt az esetet amikor egy vivőfrekvenciát használunk, adatátvitel van rajta és logikai hamis(0) jelenti a szabad vivőfrekvenciát.

Minden csúcs és él rendelkezik egy azonosítóval. A modellben egy optikai utat kétféleképp adhatunk meg:

1.)Az út mentén lévő csúcsok egymás utáni sorozataként a két végpont között

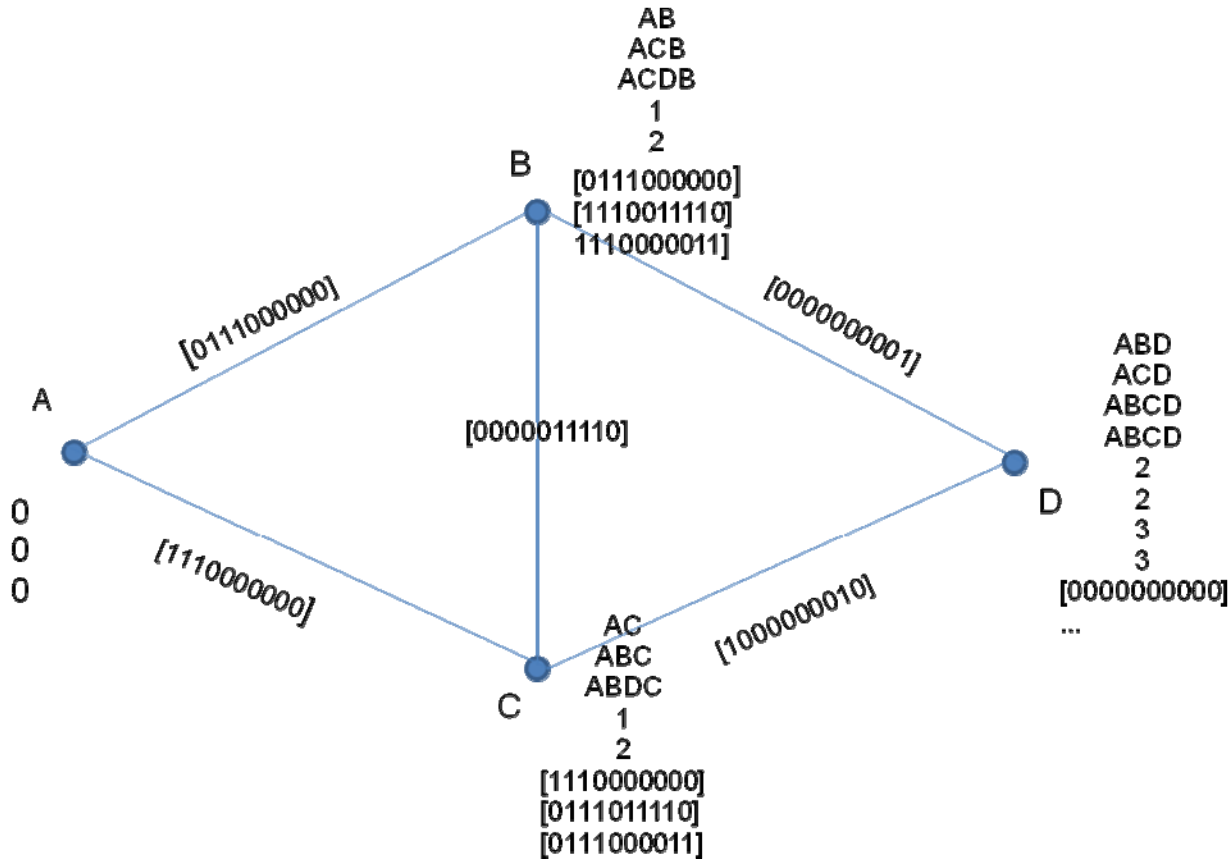
2.)Az út mentén lévő élek egymás utáni sorozataként a két végpont között

A modellben arra törekszünk, hogy egy optikai út mentén a keresztező csomópontok száma minél kisebb legyen, ami az élek számával arányos. Ezt a megfontolást azzal indokoljuk, hogy így más csomópontok között több szabad spektrum marad mintha egy rövidebb, de több központon átmenő utat választanánk. A felsorolt okok miatt egy útvonal hosszát csak az élek száma határozza meg.

Ezen modell alkalmas különböző útválasztó algoritmusok implementálására. Az általam kidolgozott és vizsgált módszerek alapját egy függvény képezi mely megadja egy csúcs szomszédjait.
Az általam kidolgozott, OFDM technológián alapuló, Spektrumfüggő legrövidebb útválasztás (Spectrum-aware Shortest Path Routing), SASP algoritmusok az optikai gerinchálózatok modellezésén alapulnak, alkalmasak útvonalválasztásra.

Szemléltetés:

Az alábbi ábrán egy a program teszteléséhez is használt mintahálózatot láthatunk:
- a kiindulópont az A pont itt nincsenek tárolt útvonalak
- az éleken egy-egy 10-es 0-1 tömb jelzi a sávszélességet ahol a 0-ák szabad vivőfrekvenciát jelölnek az 1-ek pedig foglaltat.
- a szomszédos csomópontokan látjuk a lehetséges útvonalakat és a hozzájuk tartozó lehetséges sávszélességet
- a különböző utakhoz lefoglalt frekvenciasávok között egy 0-át azaz egy OFDM vivőnyi ”guard band” (rést)–et hagyunk.



4. ábra
Szemléltető ábra

Algoritmusok

Az algoritmusokat c++ nyelven a LEMON (**L**ibrary for **E**fficient **M**odeling and **O**ptimization in **N**etworks) [6] függvénykönyvtár használatával implementáltam.
A Lemon használatával hatékony módon tudtam irányítatlan gráfokat ábrázolni. A gráf csúcsai és élei azonosítókkal rendelkeznek melyek STL vector tárolókban vannak elmentve. Az általam implementált kereső algoritmusok ezekkel az azonosítókkal dolgoznak, valamint asszociatív Map tárolóval .

Az algoritmus alapját egy rekurzív függvény adja mely mélységi bejáráson alapulva megkeresi az összes utat a gráfban.

A függvény az ’Utkereso’ osztály része , az osztály konstruktora egy Lemon formátumú gráfot és ahhoz tartozó ’Map’-et vár.

Az útkereső függvény önmagát rekurzívan hívva egy szomszédos csomópontkereső (adjecentnodes) függvény segítségével megkeresi az összes utat a-ból b-be.
Az így kapott útvonalak egész számok (csúcs vagy él azonosító) sorozata.

Adatszerkezet:

Az útvonalak kezelésére, tárolására több lehetőséget is figyelembe véve(fésűs lista,kupac) egy két dimenziós adatszerkezetet használtam (std::vector< vector<typename> >).
Ez a tároló hatékonyan kezeli az adatokat, és iterátorokat használva könnyű elérhetőséget biztosít, kevés memória művelettel. Mivel az algoritmus csak a vektor végére szúr be elemeket és sok olvasási műveletet használ ezért hatékonyabb a vektor.

A vektor előnyei:

- A szabad tárból foglal memóriát növeléskor
- Futási időben lehet a méretét csökkenteni-növelni
- Ismert a mérete nem kell számlálni
- Nem okoz memóriaszivárgást

Az optikai szálak spektrumát egy a gráf éleihez rendelt asszociatív tároló (EdgeMap) tartalmazza. Az lgf (lemon graph format) formátumú hálózat beolvasása után létrejön a ”BandWith Map” .Minden éléhez tartozik egy 0-1 sorozat mely kezdeti állapotban csupa 0 tehát a spektrum szabad.

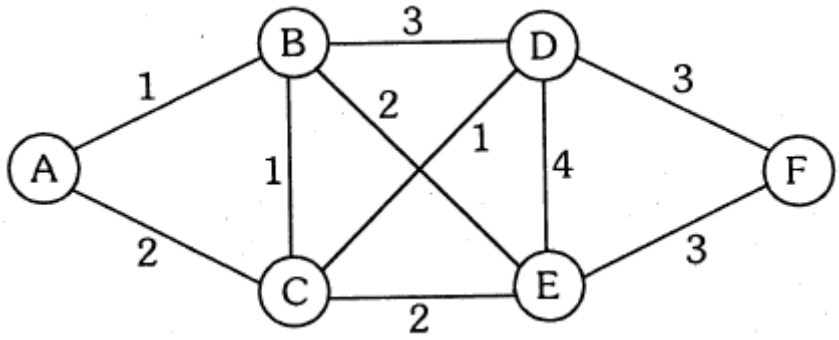
Dijkstra alapú legrövidebb útkereső (SASP-D):

Az útkereső Edsger Wybe Dijkstra algoritmusát használja.

„Az algoritmus a futása során a G gráf minden egyes v csúcspontjára nyilvántartja s csúcspont és a v közötti, a futás során addig legrövidebbnek talált út költségét. Az algoritmus indulásakor ez az érték 0 az s pontra (d[s]=0), és végtelen a G gráf minden más pontjára. Ez megfelel annak a ténynek, hogy kezdetben nem ismerünk egyetlen utat sem, ami az s pontból a többi pontba vezetne. (d[v]=∞ a V halmaz minden v elemére, kivéve s-t.) Az algoritmus befejeződésekor a d[v] az s-ből v-be vezető legrövidebb út költsége, ha létezik ilyen út - és végtelen, ha nincs ilyen út.

Az algoritmus az S és Q ponthalmazokkal dolgozik. Az S halmaz tartalmazza G gráfnak azokat a pontjait, amelyekre d[v] értéke már a legrövidebb út költségét adja meg, és a Q halmaz tartalmazza a G gráf többi csúcspontját. Az S halmaz kezdetben az üres halmaz, és az algoritmus minden egyes iterációja során egy csúcspont átkerül a Q halmazból az S halmazba. Ezt a csúcspontot úgy választjuk, hogy ennek legyen a legalacsonyabb a d[u] értéke. Amikor az u csúcspont átkerül Q-ból S-be, az összes (u,v) élre, azaz az u pont összes v szomszédjára

leellenőrzi az algoritmus, hogy az addig ismert legrövidebb utak tovább rövidíthetőek-e úgy, hogy vesszük a kiindulási ponttól az u-ig vezető legrövidebb utat és hozzáadjuk az (u,v) él költségét. Ha így kisebb költségű utat kapunk, mint az eddig ismert legrövidebb út, akkor az algoritmus a d[v] értékét ezzel az új, kisebb értékkel helyettesíti."[7]



5. ábra
Dijkstra algoritmusának szemléltetése a mintahálózaton
forrás[7]

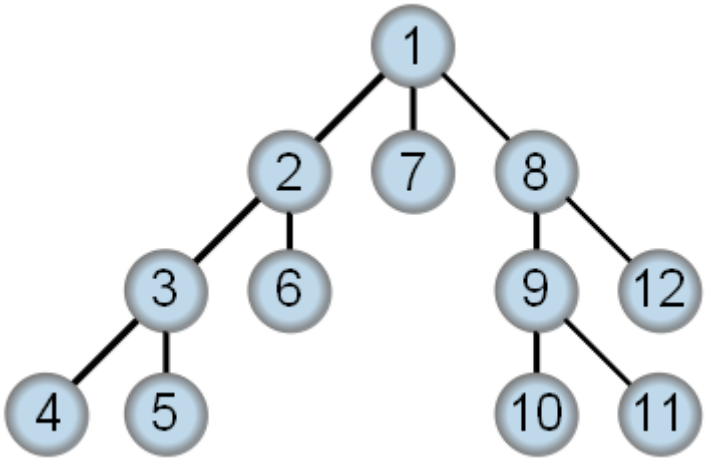
A legrövidebb út megtalálása után kapunk egy él sorozatot. Az algoritmus megnézi ,hogy lehet e az igényelt csatornaszélességet allokálni, ha nem blokkolást kapunk.

Spektrumfüggő legrövidebb útválasztás (Spectrum-aware Shortest Path Routing) módszer (SASP):

Az algoritmus működése:
Adott G(V,E) irányítatlan gráf ahol $s,t \in V$ és keressük azon $F=((v_1,v_2),(v_2,v_3),(v_3,v_4)..(v_s,v_{s+1}))$ él halmazokat melyekre $v_1,v_2,v_3..v_{s+1} \in V$ és $v_1=s, v_{s+1}=t$.
Az algoritmus mélységi keresést (depth-first search) használva keresi meg az összes hurokmentes útvonalat a két adott pont között.
Az útkereső önmagát hívva járja be a gráfot. Egy külső vektorban tárolja az éppen bejárt csúcsokat.
A kezdőponttól kiindulva az algoritmus megvizsgálja az összes kimenő élet (megkeresi az összes szomszédot) és az első talált szomszéddal hívva továbblép, a hurokmentességet biztosítása végett minden szomszédra megvizsgálja ,hogy járt e már ott.

A program így megy szintenként rekurzióval míg meg nem találja a keresett csúcsot. Ezután a keresés visszatér a legutolsó, meg nem vizsgált csúcshoz.

Az alábbi ábra jól szemlélteti egy fa gráf mélységi bejárását, a csúcsokba lévő szám azt mutatja, hogy algoritmus hányadik lépésben ér az adott csúcsba.



6. ábra
Mélységi bejárást szemléltető fa-gráf
forrás[8]

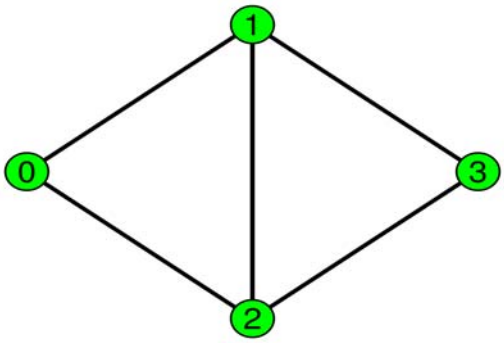
Bemenet: Egy G gráf és v csúcs

Kimenet: V-hez kapcsolódó élek és vissza élek

```
1  procedure DFS(G,v):
2      label v as explored
3      for all edges e in G.adjacentEdges(v) do
4          if edge e is unexplored then
5              w ← G.adjacentVertex(v,e)
6              if vertex w is unexplored then
7                  label e as a discovery edge
8                  recursively call DFS(G,w)
9              else
10                 label e as a back edge
```

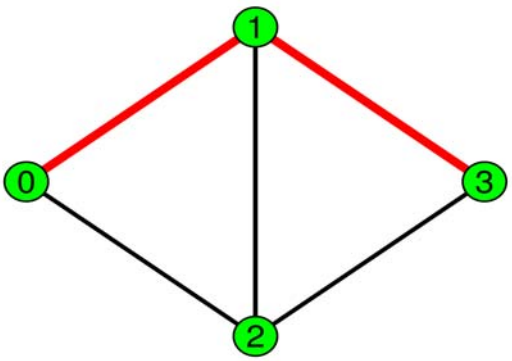
7. ábra
forrás[8]

Az összes útkereső algoritmus működését az alábbi egyszerű teszthálózaton szeretném bemutatni:



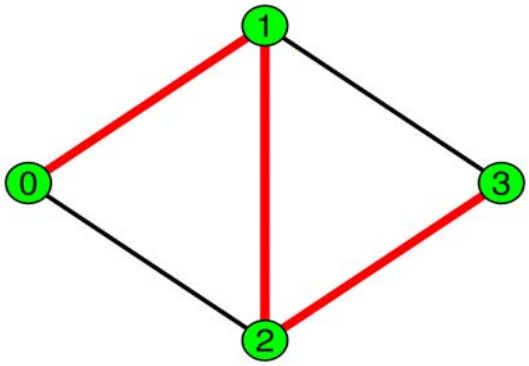
8. ábra
Egyszerűsített teszthálózat

Input: G gráf és s:=0 kezdőpont t:=3 végpont
Output: az összes lehetséges út, él vagy csomópont sorozatokként megadva
A program a fentebb részletezett mélységi kereséssel eljut a keresett pontba.
Az s kezdőpontból (0) megkeresi az s-el szomszédos csúcsokat.
Először mindig a kisebb azonosítóval rendelkező (azonosító, egész szám) csúcsot vizsgálja, ezt megjelöli bejárt ként.
Ha megtalálja a végpontot visszalép a legutoljára vizsgált ponthoz és a következő szomszédot veszi.



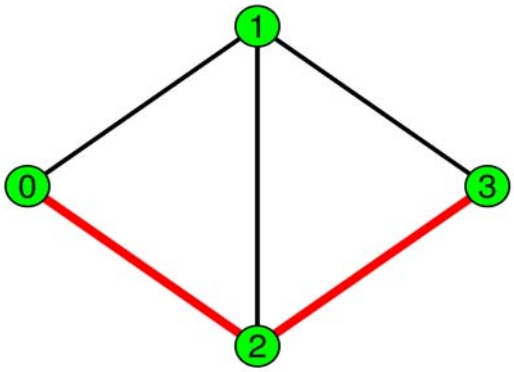
9. ábra
1. út

push(013)



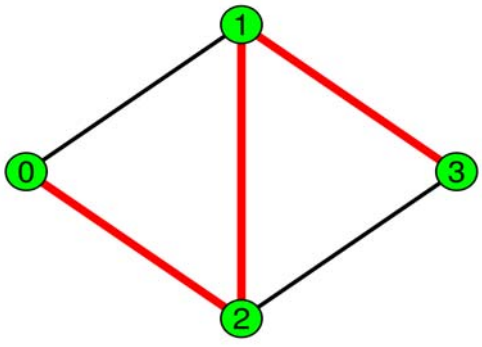
10. ábra
2. út

push(0123)



11. ábra
3.út

push(023)



12. ábra
4. út

push(0213)

Kimenet: a kimeneten az alábbi kétdimenziós adatstruktúrát kapjuk

0 1 3
0 1 2 3
0 2 3
0 2 1 3

Sávfoglalás:

Miután a program megtalálta az összes hurokmentes utat a két csúcs között és ezeket eltárolta a kétdimenziós vector-ban ezeket az utakat, hosszúság szerinti növekvő sorba rendezi.
A sorba rendezés kupacrendező algoritmussal történik mivel a legrosszabb esetben is N*logN lépést hajt végre ahol N az elemek száma.
Az így nyert adatszerkezetet a legrövidebb úttól kezdve vizsgáljuk.
Ha egy x szélességű frekvenciasávra érkezik kérés, akkor x+2 széles szabad sávot keresünk, mivel két optikai út között a spektrumban kell, hogy legyen egy szabadon hagyott védősáv. Ebben a modellben a védősávot egy vivőfrekvencia szélességűre választottuk.

Az x+2 széles szabad sávnak a teljes út minden élén teljesülni kell.
Ha a vizsgált útra teljesül a feltétel akkor lefoglaljuk a megfelelő éleken az x-széles spektrum sávot.
Ha nem teljesül, vizsgáljuk a rendezett utak közül a következőt, azaz a második legrövidebbet.

Spektrumfüggő legrövidebb útválasztás (Spectrum-aware Shortest Path Routing) módszer két irányú csatornafoglalással(SASP-2):

A futási idő csökkentése és a jobb spektrumkihasználás érdekében módosítottam az előző algoritmust.
Ez az algoritmus két irányból vizsgálja egy optikai út mentén a spektrumot. Az út mentén lévő élek spektrumának logikai vagyolása után az optikai út teljes hosszában szabad csatornákat kapjuk. Egy ilyen spektrumot vizsgál az algoritmus egyszerre két irányból kezdve, az alacsonyabb frekvenciájú csatornáktól a magasabb felé és vissza legnagyobb frekvenciától a kisebbek felé. Ahol előbb talál elég széles szabad sávot ott allokál.

Szimuláció

A szimulációhoz a LEMON (Library for Efficient Modeling and Optimization in Networks) véletlenszám generátorát használva generáltam kéréseket véletlenszerű két pont között. A kérések száma az aktuális hálózat bonyolultságától, nagyságától függ.
Egy optikai szálon az OFDM csatornák, vivőfrekvenciák számát 30-nak választottam.
A forgalmi kérések csatornaigénye véletlenszerűen változik 1 és 5 között.
A szimuláció célja megmutatni, hogy adott körülmények között (túlterhelt hálózat vagy ideális eset) melyik algoritmus tudja kiszolgálni a legtöbb kérést.

Előzetes feltevések:

Várakozásaim szerint a Dijkstra algoritmust alkalmazó legrövidebb útválasztó algoritmus lesz a leggyorsabb, és ideális terhelés esetén a legtöbb blokkolást is ennél a módszernél tapasztalhatjuk. Ez az algoritmus már abban az esetben is blokkolással tér vissza, ha a két pont közötti legrövidebb út egy élén nincs elég szabad csatorna. Viszont sikeres allokálás esetén ez a módszer használja a legkevesebb erőforrást, hiszen mindig a legrövidebb út mentén foglal le erőforrásokat.

Az SASP módszert használó algoritmus hosszabb ideig fog futni, viszont később lesz az első blokkolás. Az algoritmus abban az esetben, amikor a legrövidebb út mentén blokkolás van további útvonalakat vizsgál, és ha lehetséges egy hosszabb élsorozaton foglal sávot.
A hálózat normális terhelése mellett tehát várhatóan kevesebb blokkolással fog teljesíteni ez a módszer, viszont több erőforrást használ.
A hálózat túlterhelt állapotában több blokkolást fogunk kapni mint a Dijkstra algoritmust használva.

A módosított SASP módszer várhatóan szimmetrikusabb spektrumeloszlást eredményez.

A k-Shortest megközelítés átmenet a SASP módszer és a legrövidebb út módszer között. Viszont az algoritmus csak a legrövidebb út hosszának függvényében ad meg utakat, ezért hamarabb kapunk blokkolást.

Mérendő adatok:

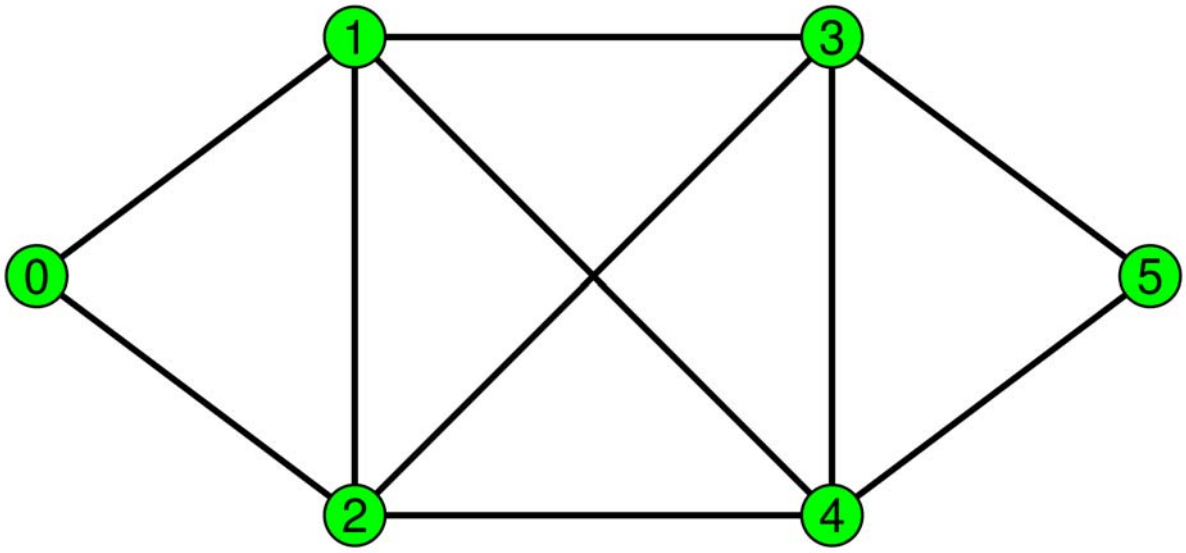
Az algoritmusok hatékonyságát legjobban a sikeresen teljesített kérések számával lehetne jellemezni.

Feltételezzük ,hogy a hálózat normál állapotban(áteresztőképességének megfelelő terhelés alatt) minden forgalmi igényt ki kell ,hogy szolgáljon. Ezen oknál fogva fontos mérőszám lehet, hogy hány kérést tud kiszolgálni blokkolás nélkül, azaz hány teljesített kérés van az első blokkolásig.

További érdekes adat lehet a spektrum telítettsége. Összes szabad csatorna és összes foglalt csatorna aránya.

Futtatás a tesztelő hálózaton

A teszteléseket az alábbi hálózaton kezdtem, az egy élen lévő szabad csatornák számát 10-nek választva, itt könnyen ellenőrizhető volt a helyes működés:



13. ábra
Próba hálózat

A program a csomópontokban található (integer) azonosítókkal dolgozik.

Teszthálózat: 5 10 20 40 80 kérésre futtatva
Az eredmények az igényelt csatornák száma szerint : véletlenszerű 1-3,3,2,1

	SASP-D	SASP
Blokkolások száma	0,0,0,0	0,0,0,0

5

	SASP-D	SASP
Blokkolások száma	0,0,0,0	0,0,0,0

10

	SASP-D	SASP
Blokkolások száma	3,6,3,0	1,7,1,0

20

	SASP-D	SASP
Blokkolások száma	12,18,12,3	11,19,11,0

40

	SASP-D	SASP
Blokkolások száma	38,49,43,29	41,51,42,29

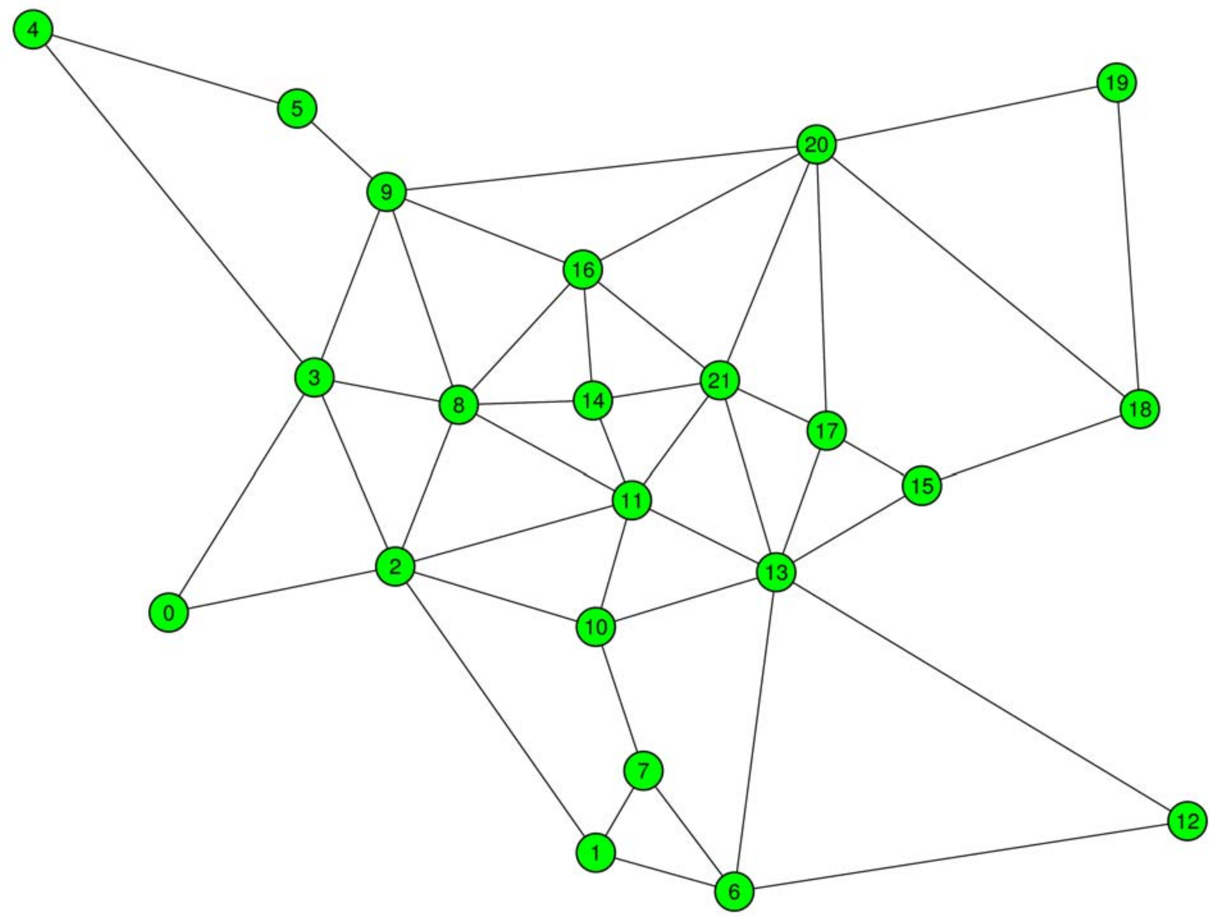
80

Első blokkolásig teljesített kérések kis a hálózaton:

	SASP-D	SASP
1 csatorna	27	32
2 csatorna	10	16
3 csatorna	5	10

Az eredményekből láthatjuk, hogy jelentősen később kapjuk az első blokkolást az összes út módszernél, viszont túlterhelt hálózat esetén a Dijkstra alapú módszer jobb.

Futtatás a 22 csomópontos európai hálózaton



14. ábra

a 22 csomópontos európai teszthálózat

Csomópontok száma: 22

Élek száma: 45

A 14.ábrán lévő hálózatot a LEMON egy függvényével rajzoltam ki eps formátumú képbe. A csomópontokban látható számok a program által is használt azonosító értékek.

Blokkolások(3 csatorna igény)

SASP:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 3 4 4 4 5 6 6 6 6 6 6 7 7 8 8 9 9

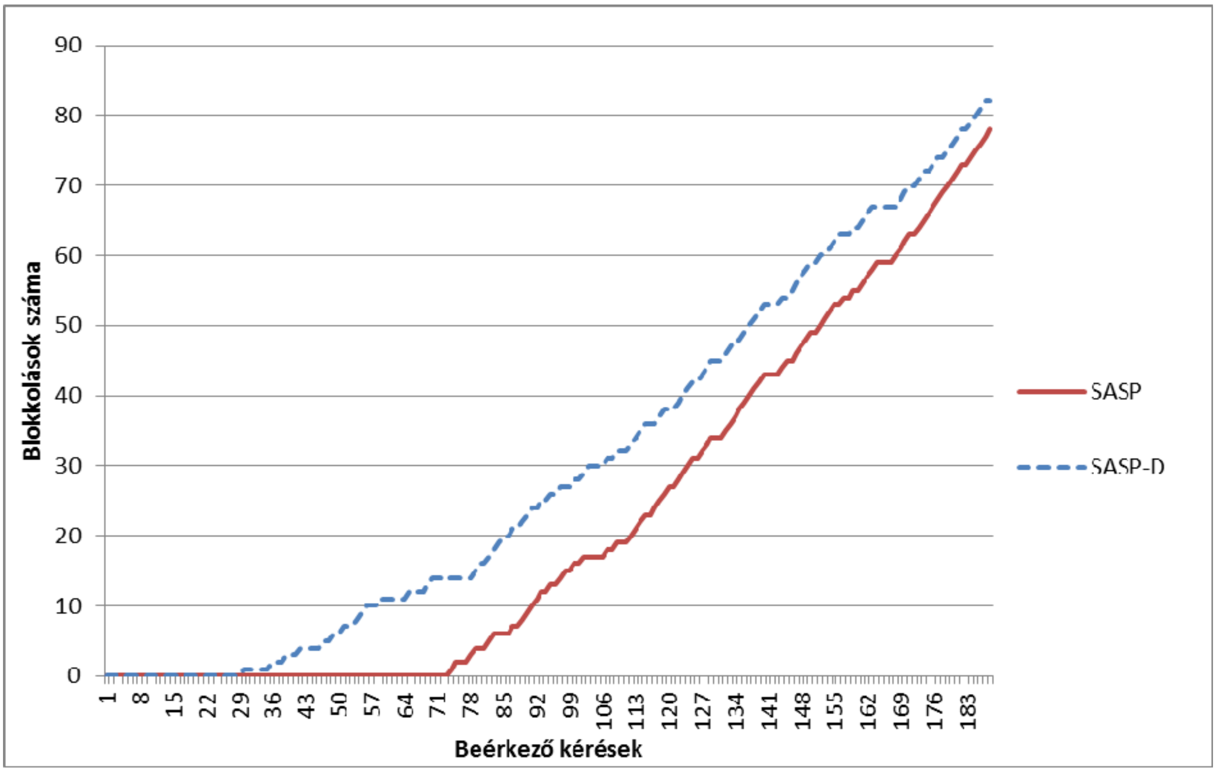
SASP-D: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 2 2 2 3 3 3 4 4 5 5 6 6 6 6 6 7 8 9 9 9 10 10 10 10 10 10 11 11 11 11 12 13 13 13 13 13 13 13 13 13 14 15 15 16 17 17 18 18 19 19 20 21 22 22 23 23

SASP-2: 53 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 3 4 4 4 5 6 6 6 6 7 7 8 8 9 9 10 10

Eredmények kiértékelése:

A alábbi (15.ábra) ábrán látható, hogy kezdetben az összes utat kereső algoritmus jobban teljesít, később következik be az első blokkolás és összességében is kevesebb blokkolás van. A Dijkstra módszer nem tudja kezelni a legrövidebb út blokkolása esetén fennálló problémát, ellenben az összes út módszer ilyenkor is talál utat viszont több erőforrást foglal le. Ebből következik a görbe további menete, ahogy kezd telítődni a spektrum egyre kevesebb szabad sávot talál az összes út módszer, viszont a Dijkstra algoritmus alapú legrövidebb útkereső nem foglalt le korábban több erőforrást.

Ahogy nő a kérések száma, egyre közelít egymáshoz a két görbe.



15. ábra

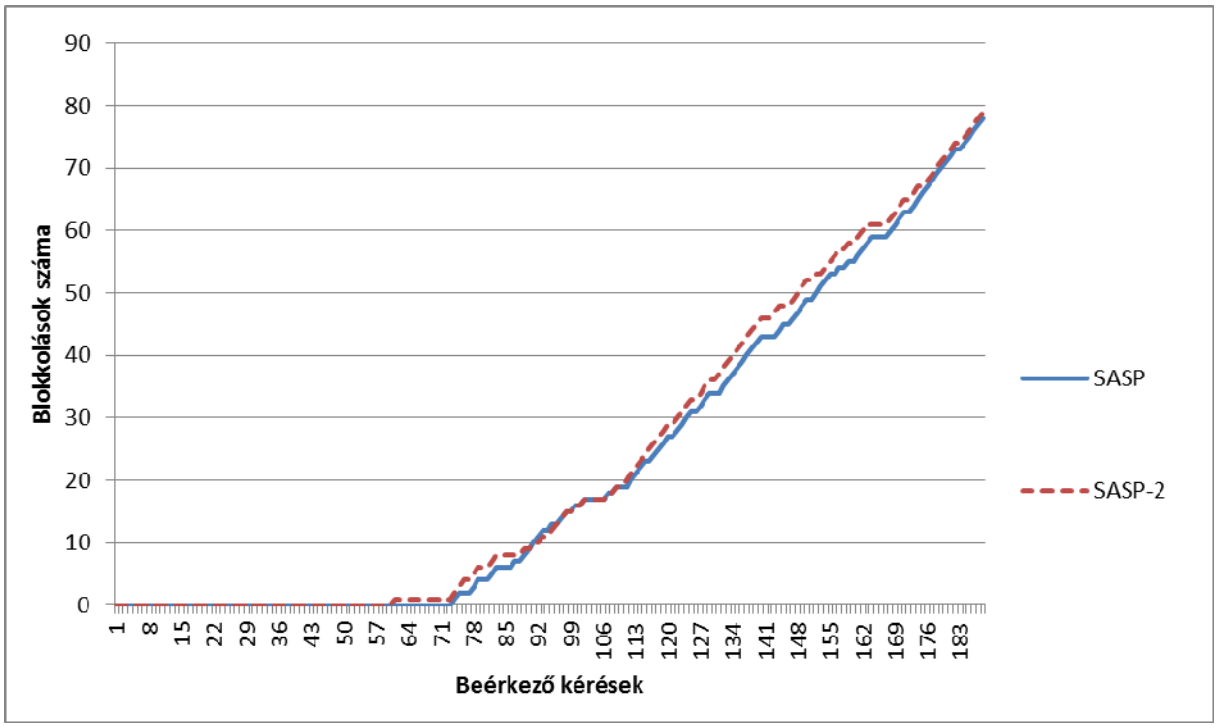
SASP és SASP-D 200 kéréssel az európai hálózaton

Összes blokkolások száma: Dijkstra(SASP-D) 82

Összes út módszer(SASP) 78

Teljesített kérések az első blokkolásig: Dijkstra (SASP-D) 29

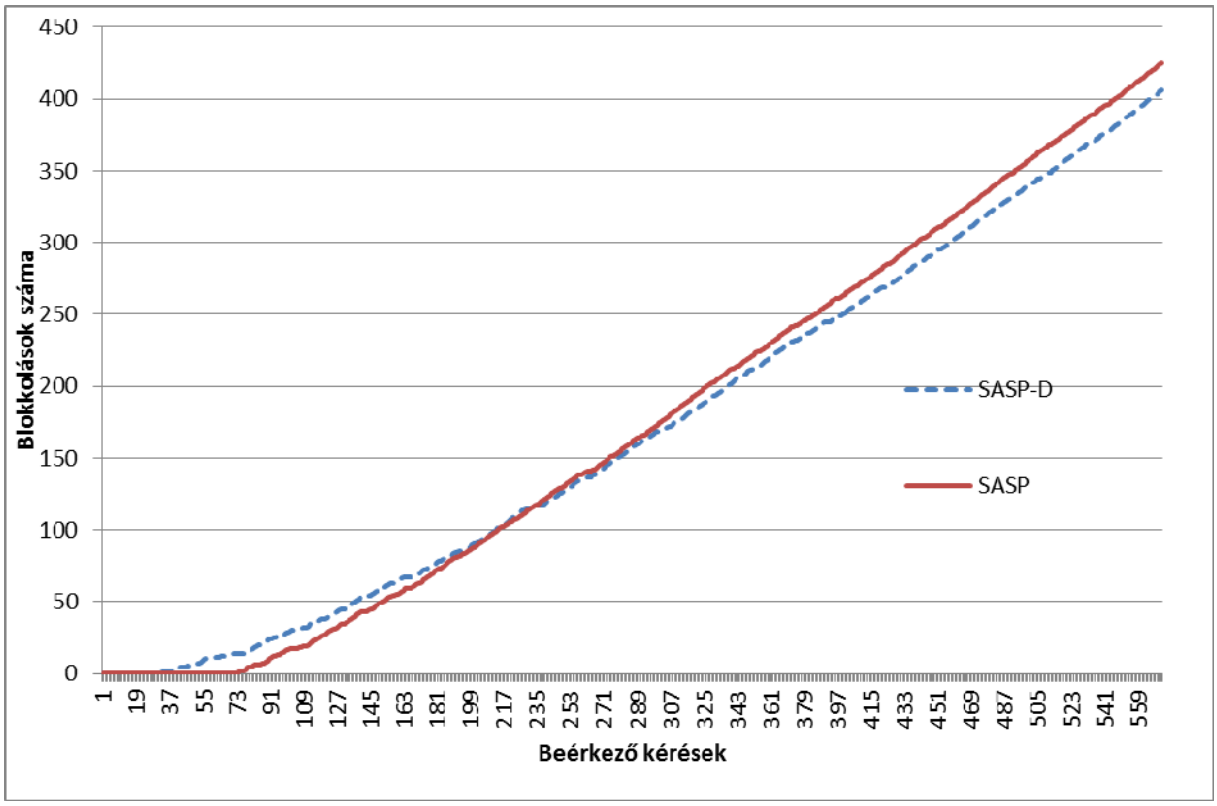
Összes út módszer(SASP) 73



16. ábra

Az egyirányú és kétirányú spektrumkitöltés vizsgálata blokkolások szempontjából 200 kérés

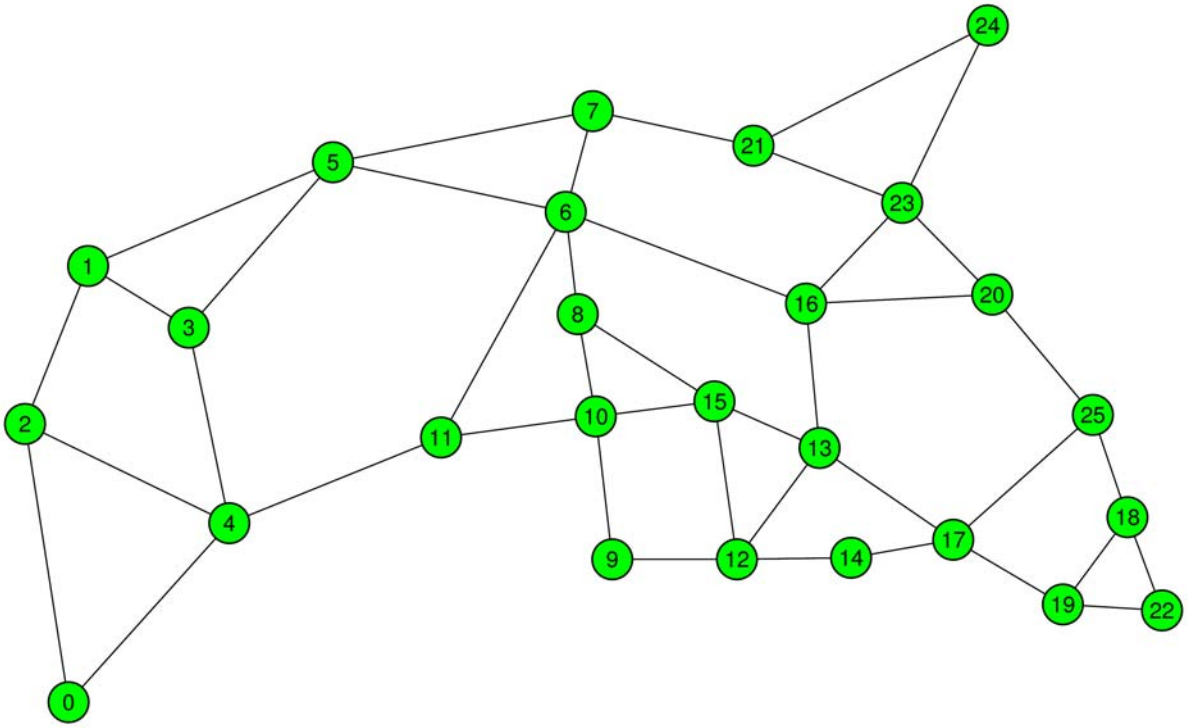
Az alábbi grafikonon (17. ábra) látható az a pont ahol a két görbe metszi egymást ez az állapot a 22 pontos európai hálózaton tesztelve a 200.ik beérkező kérés közelében van. Amikor a hálózat terheltsége ezen szint fölé emelkedik hatékonyabb a legrövidebb utat választó módszer, mivel ilyen terheltség mellett átlagosan több utat tud elvezetni.



17. ábra

A blokkolások alakulása a kérések függvényében 600 kérés esetén

Futtatás a 26 csomópontos észak-amerikai hálózaton

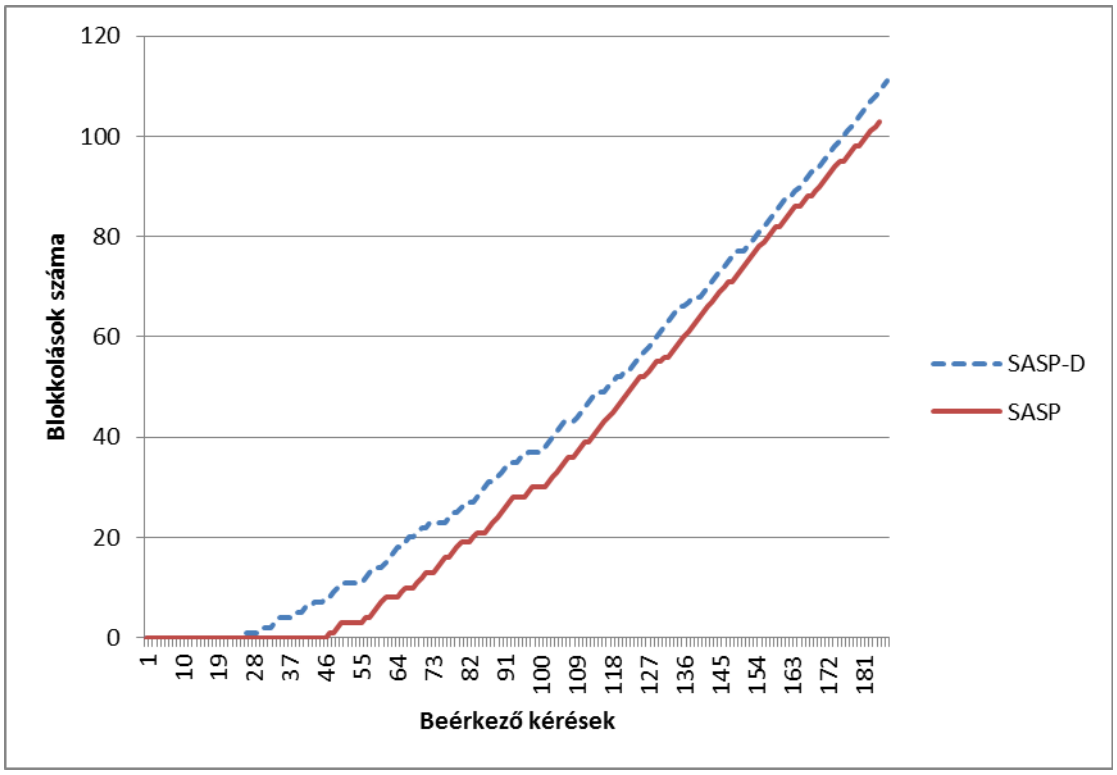


18. ábra
26 pontos észak-amerikai hálózat

A fentebbi ábrán (18.ábra) lévő hálózatot a LEMON egy függvényével rajzoltam ki eps formátumú képbe. A csomópontokban látható számok a program által is használt azonosító értékek.

Csomópontok száma: 26

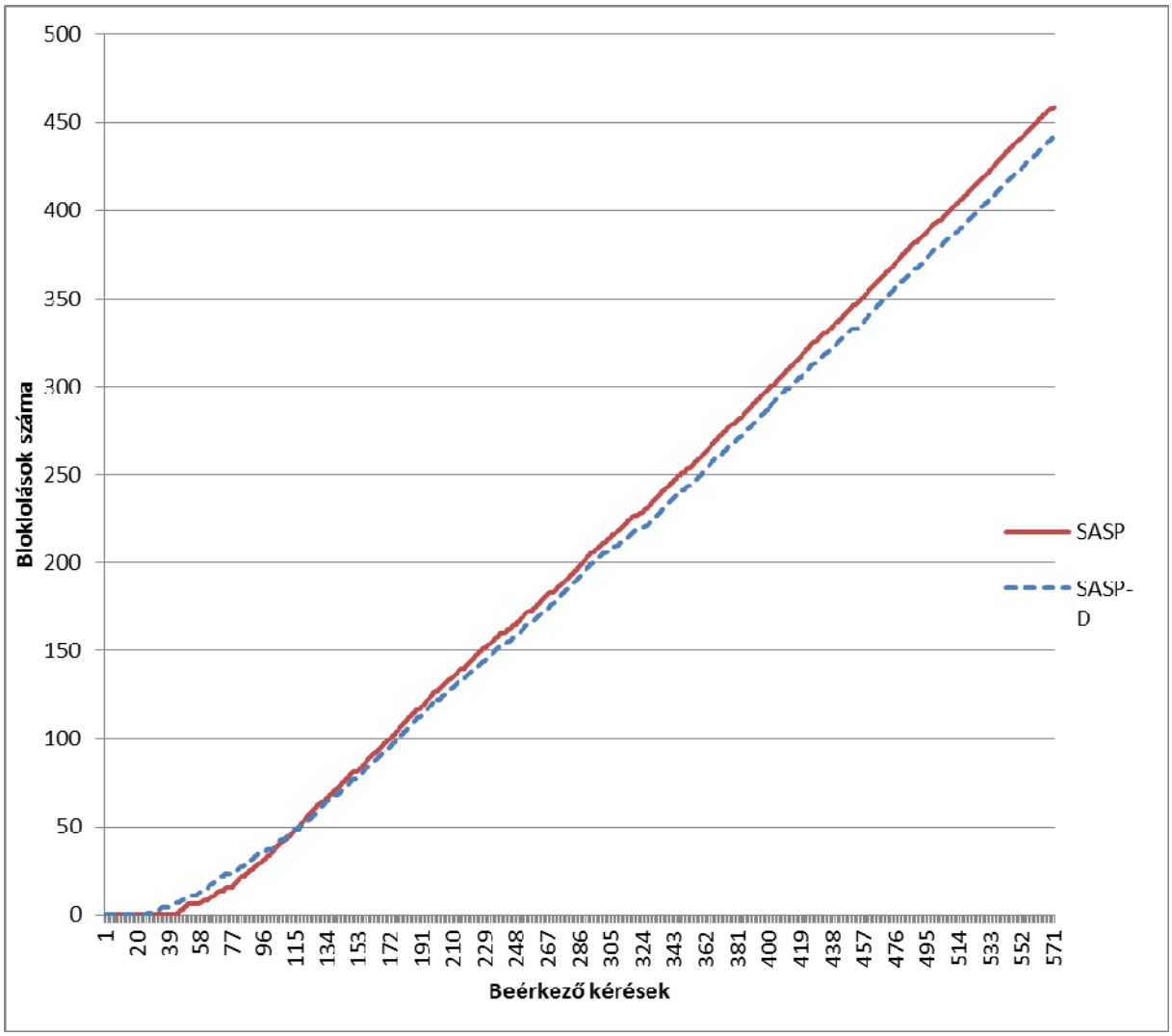
Élek száma: 42



19. ábra

Dijkstra algoritmus és az összes út módszer 200 kérés esetén

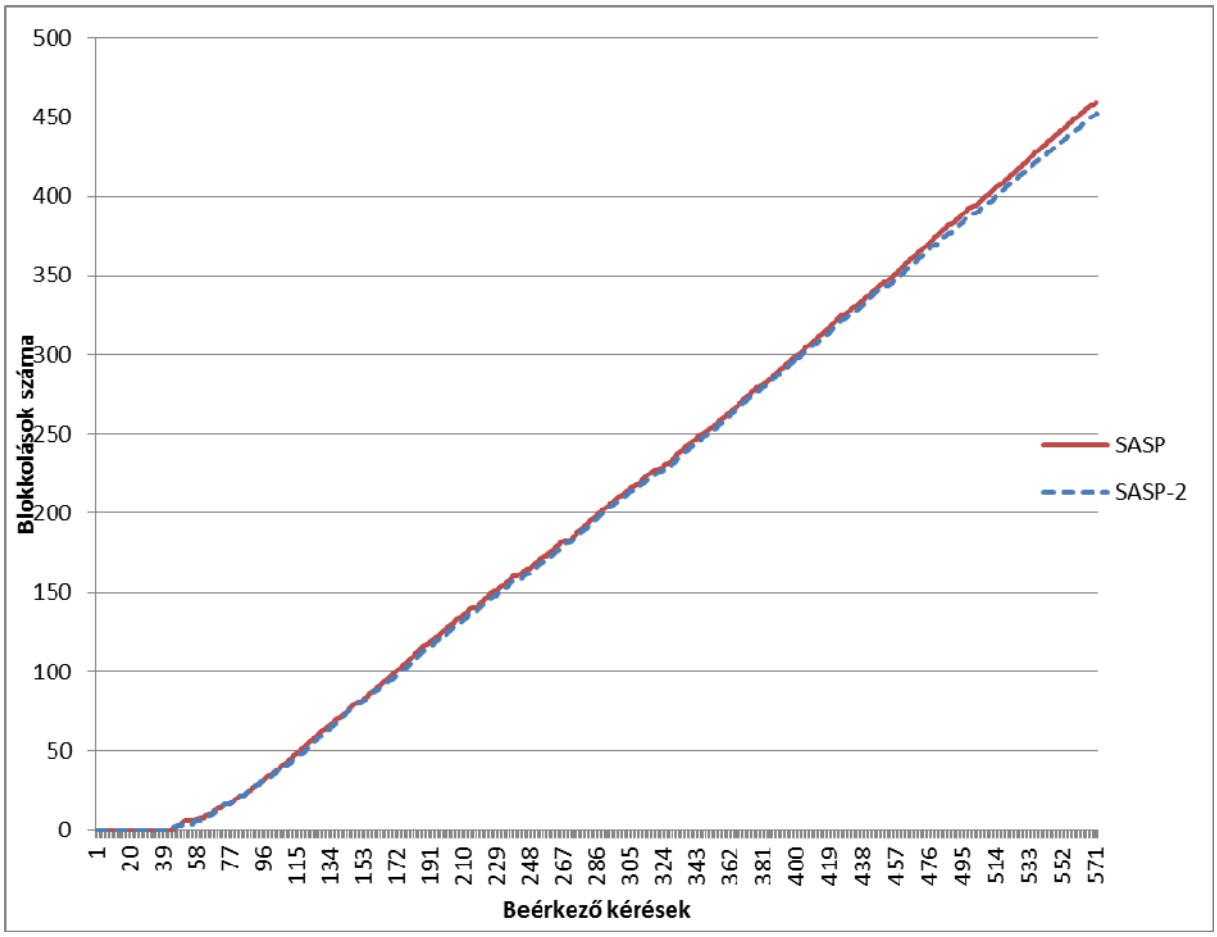
A 19-es ábrán láthatjuk ,hogy később érkezik az első blokkolás az összes út módszert használó algoritmust futtatva. Ahogy emelkedik a kérések száma közelít a két módszer hatékonysága.



20. ábra

A Dijkstra algoritmus és az összes út módszer 600 kérés esetén

Az európai hálózaton végezett tesztehez(17.ábra) hasonlóan itt is(20.ábra) metszi egymást a két görbe. Itt a metszéspont 115 kérésnél van.
Túlterhelt hálózat esetén a Dijkstra algoritmus alapú SASP-D itt is jobban teljesít.



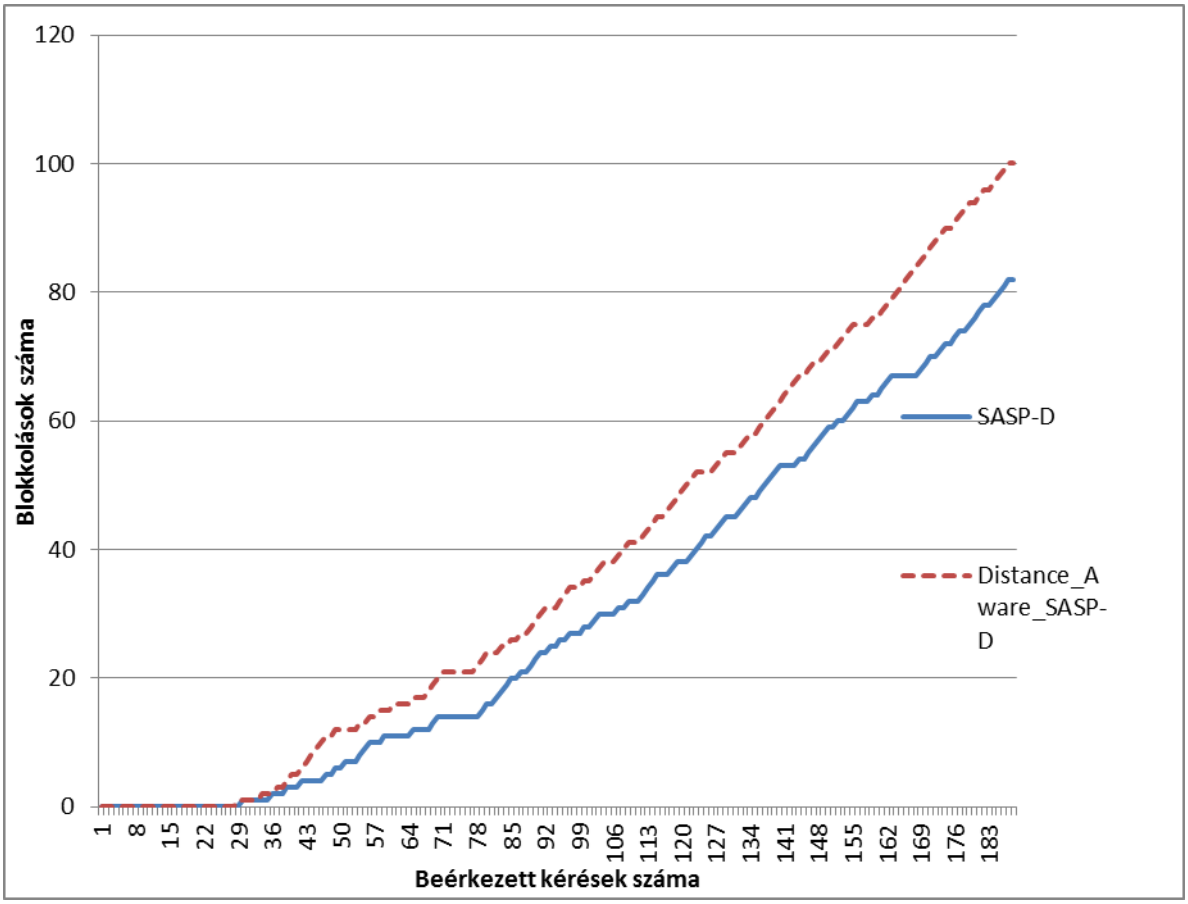
21. ábra

Az összes út módszer és a kétirányú összes út módszer 600 kérés esetén

Távolságfüggő modell

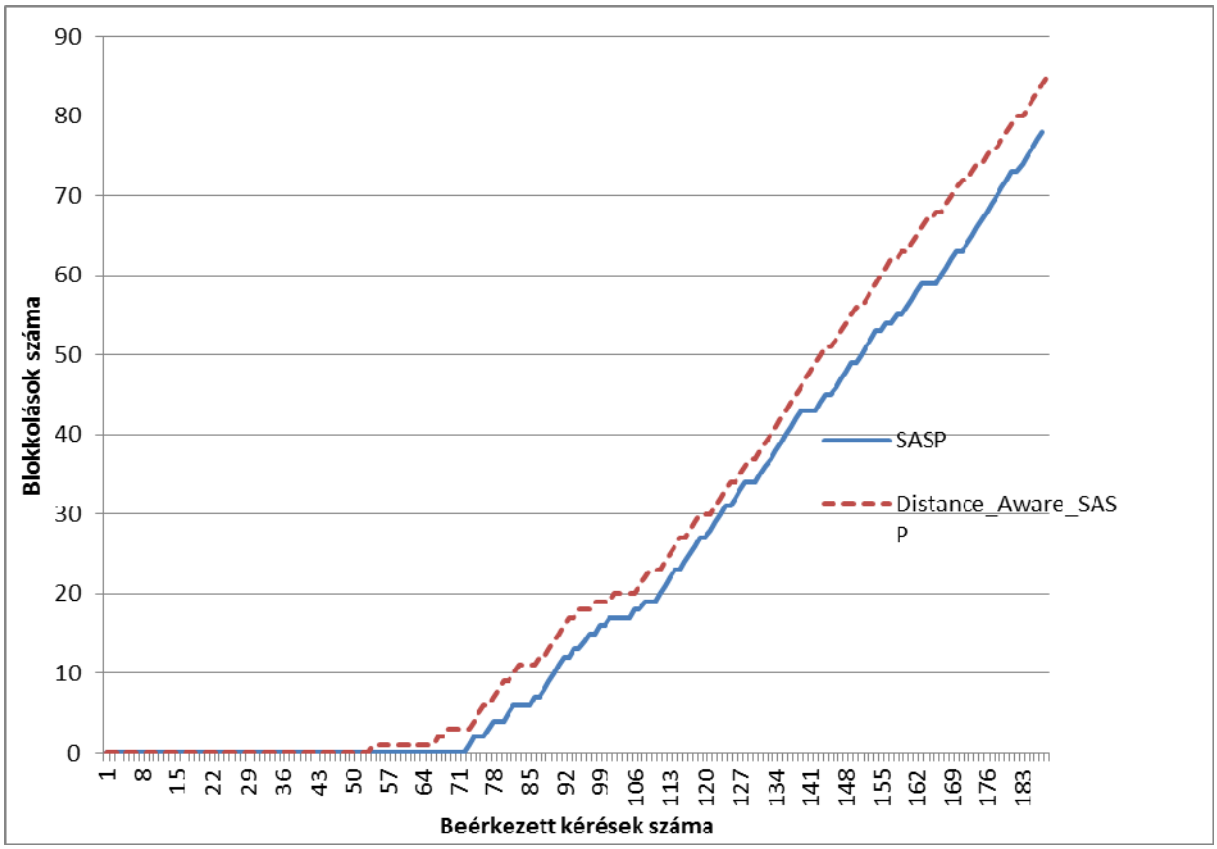
Az algoritmusokat teszteltem olyan modell esetén is ahol egy optikai út hosszát nem a benne lévő élek és csomópontok száma adja, hanem az élekhez rendelt hosszak összege. Az európai és észak-amerikai hálózatokban az élekhez adva van egy költség, amit eddig nem vettem figyelembe. A költség jelentheti a két szomszédos központ közötti távolságot, így célunk lehet az is, hogy minél rövidebb optikai szálakon továbbítsuk a forgalmi igényt.

A 22.ábrán összevetettem a költség alapú és az egység él hosszúságon alapuló algoritmusokat. A programokat kipróbáltam az európai és az észak-amerikai hálózatra is, hasonló eredményekkel.



22. ábra

A két Dijkstra algoritmus a 22 pontos európai hálózaton



23. ábra

A két összesút kereső algoritmus a 22 pontos európai hálózaton

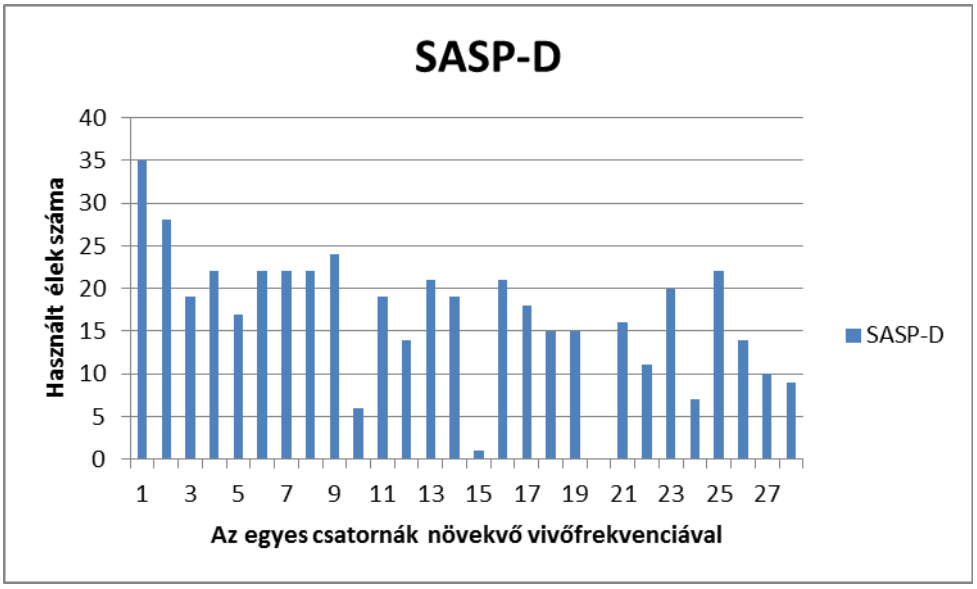
Látható, hogy mindkét esetben kevesebb blokkolást kapunk az eredeti modellel, ami nem veszi figyelembe az optikai szálak tényleges hosszát.

Az élhosszakat figyelembe vevő algoritmus sok esetben hosszabbnak találhat egy kevesebb élből álló utat. Ilyenkor az eredeti módszer kevesebb élen vezeti el a forgalmat több szabad spektrumot hagyva a hálózatban.

Energia hatékonyság szempontjából is jobb, ha kevesebb központon haladunk át, viszont hosszabb optikai szálon nagyobb valószínűséggel keletkezhet hiba.

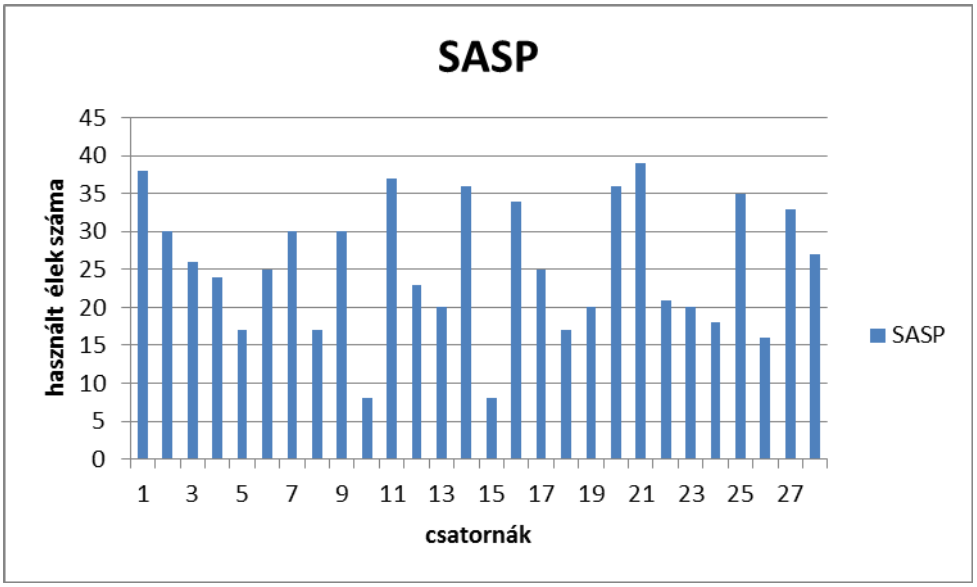
A forgalom eloszlása az egyes csatornákon

Az alábbi grafikonok (24-27.ábrák) a forgalom eloszlását mutatják az egyes csatornákon. Balról jobbra növekszik az OFDM vivők frekvenciája.



24. ábra

Az egyes élek terheltségének eloszlása a hálózatban Dijkstra futtatása után

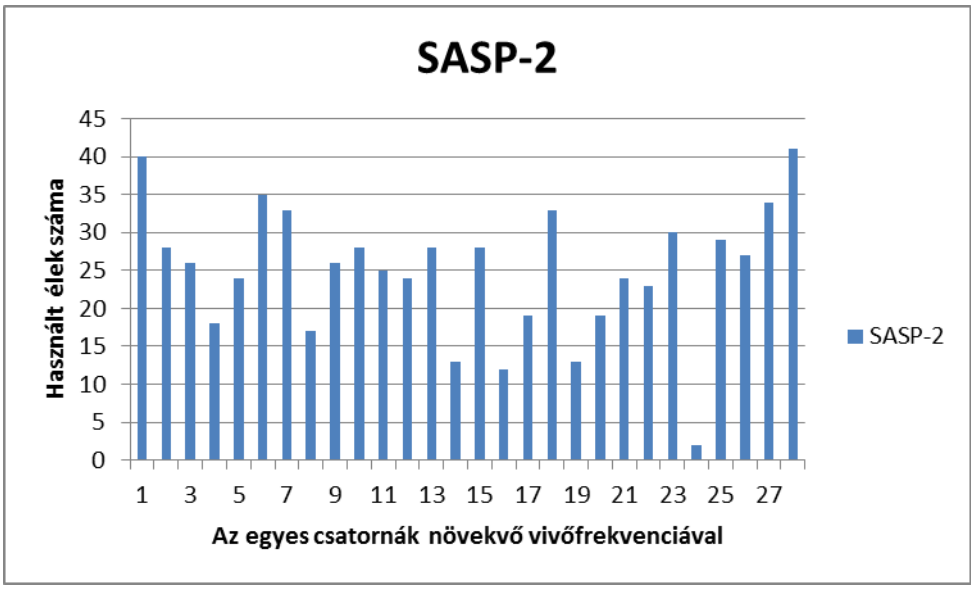


25. ábra

Az egyes élek terheltségének eloszlása a hálózatban AllPath futtatása után

Az ábrákból több következtetés is levonható:
A fentebbi két ábrán jól látható, hogy 200 beérkező kérés esetén melyből a Dijkstra algoritmussal 118-at míg az összes út módszerrel 122-őt szolgáltunk ki sokkal telítettebb a háózat a második esetben.

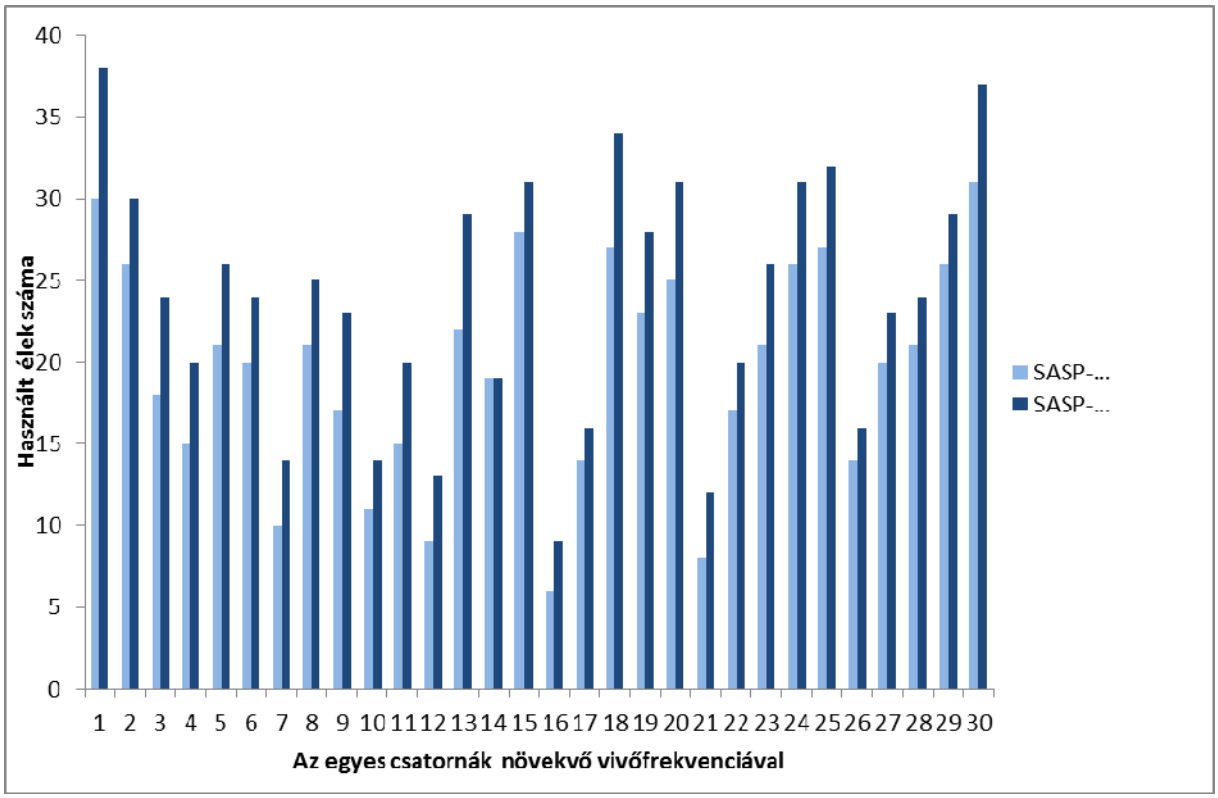
Az összes út algoritmus később okoz blokkolást viszont több erőforrást használ az elkerülő hosszabb utak miatt.



26. ábra

Az egyes élek terheltségének eloszlása a hálózatban kétirányú AllPath futtatása után

A kétirányú spektrumkitöltést alkalmazvaegyszerre vizsgáljuk az adott út mentén az élek spektrumát az alacsonyabb ferkvenciáktól fölfelé és a legmagasabb vivőfrekvenciájú csatornáktól lefelé. Ezzel a módszerrel sokkal szimetrikusabb képet kapunk.
Látni, hogy a spektrum perem nagyobb az élek használtságának aránya.



27. ábra
Az optikai szálak terheltsége 200 és 600 beérkező kérés esetén

Futási idő:

SASP= 22_eu,200kérés(rnd1-5),30 csatorna 6:18

SASP-2= 22_eu,200kérés(rnd1-5),30 csatorna 4:15

A kétirányú spektrumfoglalás esetén átlagosan 30%-al csökkent a futási idő.
A Dijkstra algoritmus algoritmus futási ideje mindig 1 sec alatt maradt.

A SASP algoritmus eredményei alapján a futási idő és komplexitás csökkentése érdekében egy új módszer kidolgozásába kezdtem mely nem veszi figyelembe az összes lehetséges optikai utat a két pont között, hanem egy előre megadott k számú legrövidebb út algoritmussal dolgozik, bár így egyes esetekben nő a blokkolás valószínűsége.

Algoritmusok bonyolultsága:

Az összes út keresési algoritmus NP-teljes probléma.
A k-Shortest Path módszer megoldható polinom időben(lásd. Eppstein algoritmusa).

Megjegyzés:

A tesztelő hálózaton futtatva sokkal hamarabb jelenkezetek a Dijkstra alapú és az összes útkereső algoritmus közti különbségek. A terhelés növekedésével a SASP-D jelentősen kevesebb blokkolást okozott.

Az európai és amerikai hálózatokon nem mutatkozott ekkor különbség, ennek oka, hogy a tesztelő hálózaton a véletlenszerűen kiválasztott csúcsok által kapott utak hossza nagyobb szórást mutat és a SASP algoritmus már hamar a legrövidebb útnál többszörös hosszúságú utat talál, ezzel arányosan több erőforrást foglal le, míg a nagyobb hálózatok esetén a SASP által első szabad legrövidebb út és a Dijkstra alapú algoritmus által talált legrövidebb út hossza között nincs olyan nagy különbség.

Érdekes megemlíteni, hogy az észak-amerikai hálózaton futtatva a SASP mintegy ötöd annyi idő alatt futott le. A hasonló nagyságú hálózatokon tapasztalt ilyen mértékű eltérés a futási időben a hálózatot eltérő topológiájával magyarázható.

Összegzés

Tudomásunk szerint még nem születtek algoritmusok az O-OFDM modulációs technikán alapuló optikai jelátvitel területén, melyek az OFDM adta előnyöket kihasználva, futás közben a szabad spektrumot figyelembe véve valósítanak meg útvonalválasztást.
Dolgozatomban kifejtettem az általam kidolgozott három megoldás működését és valós optikai gerinchálózatok modelljein futtatott szimulációkkal támasztottam alá hatékonyságukat.
Az eredményeket tekintve minden esetben jobbnak bizonyult az a modell melyben az optikai szálak egységnyi hosszúak következésképp az utak hossza csak az élek számától függ.
Az algoritmusokat összevetve a különböző helyzetekben,a hálózat normál terhelése mellett, az összes utat kereső algoritmust(SASP,SASP-2) ajánljuk.
A hálózat túlterhelése esetén kevesebb blokkolást kaptam a Dijkstra algoritmus alapú legrövidebb útkereső algoritmust használva. Az utóbbi körülmények között a SASP-D algoritmust javasoljuk.
A futási időket is figyelembe véve a SASP-D jelentősen gyorsabb(1 sec alatt) a SASP és SASP-2-nél.
A SASP-2-vel az egyszerre történő kétirányú keresés használatával átlagosan 30%-os futási idő csökkenést értem el.
Egy hatékonyságában SASP-2 vel egyező de futási idejét tekintve nagyságrendileg gyorsabb algoritmus kidolgozásába kezdtem.

Irodalomjegyzék

1. A Survey on OFDM-Based Elastic Core Optical Networking: Guoying Zhang, Marc De Leenheer, *Member, IEEE,* Annalisa Morea, *Member, IEEE,* and Biswanath Mukherjee, *Fellow, IEEE;*

2. Spectrum-Efficient and Scalable Elastic Optical Path Network: Architecture, Benefits, and Enabling Technologies: Masahiko Jinno, Hidehiko Takara, Bartlomiej Kozicki, Yukio Tsukishima, Yoshiaki Sone, and Shinji Matsuoka, NTT Corporation

3. Traffic Grooming in Spectrum-Elastic Optical Path Networks :Yi Zhang,Xiaoping Zheng, Qingshan Li, Nan Hua, Yanhe Li, and Hanyi ZhangTsinghua National Laboratory for Information Science and Technology Department of Electronic Engineering, Tsinghua University, Beijing, 100084, China

4. Highly Survivable Restoration Scheme Employing Optical Bandwidth Squeezing in Spectrum-Sliced Elastic Optical Path (SLICE) Network: (Yoshiaki Sone*, Atsushi Watanabe*, Wataru Imajuku+, Yukio Tsukishima*, Bartlomiej Kozicki*,Hidehiko Takara*, and Masahiko Jinno*)

5. Distance-Adaptive Spectrum ResourceAllocation in Spectrum-Sliced Elastic Optical Path Network: Masahiko Jinno, Bartlomiej Kozicki, Hidehiko Takara, Atsushi Watanabe, Yoshiaki Sone, TakafumiTanaka, and Akira Hirano, NTT Corporation

6. Lemon (Library for Efficient Modeling and Optimatization in Networks) http://lemon.cs.elte.hu

7. Dijkstra's algorithm http://en.wikipedia.org/wiki/Dijkstra's_algorithm

8. Depth-first search http://en.wikipedia.org/wiki/Depth-first_search

9. Wavelength-division multiplexing http://en.wikipedia.org/wiki/Wavelength-division_multiplexing

10. Finding the k Shortest Paths: David Eppstein 35th Annual Symposium on Foundations of Computer Science, 1994 Proceedings.,

Ábrajegyzék

Függelék:

Bemenő adat file:

A 22 csomópontos európai hálózat:

```
nodes
label coords
Ebben a részben láthatjuk a csomópontok adatait:
-első oszlopban a csomópontok azonosítója
-második oszlopban a csomópontok koordinátája
0     (94,194)
1     (390,27)
2     (251,226)
3     (195,357)
4     (0,599)
5     (183,544)
6     (486,0)
7     (423,84)
8     (295,338)
9     (245,486)
10    (390,184)
11    (415,272)
12    (800,49)
13    (515,222)
14    (388,341)
15    (616,282)
16    (381,432)
17    (550,320)
18    (767,335)
19    (751,562)
20    (543,519)
21    (476,355)
@edges
Alább az élek adatait láthatjuk:
-első két oszlopban van megadva, hogy mely két csomópont között vannk
-harmadik oszlopban található az egyedi azonosítójuk
-negyedik oszlopban a párhuzamos élek szám
-utolsó oszlopban a költség(távolság)
           label cap   cost
0     2     0     1     19
0     3     1     1     22
1     2     2     1     18
1     6     3     1     9
1     7     4     1     19
2     0     5     1     19
2     1     6     1     18
2     3     7     1     6
2     8     8     1     19
2     10    9     1     9
2     11    10    1     12
3     0     11    1     22
3     2     12    1     6
3     4     13    1     18
3     8     14    1     10
3     9     15    1     10
```

```
4     3     16    1     18
4     5     17    1     16
5     4     18    1     16
5     9     19    1     17
6     1     20    1     9
6     7     21    1     20
6     12    22    1     21
6     13    23    1     16
7     1     24    1     19
7     6     25    1     20
7     10    26    1     23
8     2     27    1     19
8     3     28    1     10
8     9     29    1     21
8     11    30    1     7
8     14    31    1     19
8     16    32    1     21
9     3     33    1     10
9     5     34    1     17
9     8     35    1     21
9     16    36    1     8
9     20    37    1     18
10    2     38    1     9
10    7     39    1     23
10    11    40    1     12
10    13    41    1     11
11    2     42    1     12
11    8     43    1     7
11    10    44    1     12
11    13    45    1     17
11    14    46    1     17
11    21    47    1     22
12    6     48    1     21
12    13    49    1     19
13    6     50    1     16
13    10    51    1     11
13    11    52    1     17
13    12    53    1     19
13    15    54    1     20
13    17    55    1     15
13    21    56    1     16
14    8     57    1     19
14    11    58    1     17
14    16    59    1     21
14    21    60    1     21
15    13    61    1     20
15    17    62    1     20
15    18    63    1     17
16    8     64    1     21
16    9     65    1     8
16    14    66    1     21
16    20    67    1     22
16    21    68    1     21
17    13    69    1     15
17    15    70    1     20
17    20    71    1     21
17    21    72    1     7
18    15    73    1     17
```

```
18    19    74    1     7
18    20    75    1     13
19    18    76    1     7
19    20    77    1     19
20    9     78    1     18
20    16    79    1     22
20    17    80    1     21
20    18    81    1     13
20    19    82    1     19
20    21    83    1     8
21    11    84    1     22
21    13    85    1     16
21    14    86    1     21
21    16    87    1     21
21    17    88    1     7
21    20    89    1     8
@attributes
avg_distance 2.46753
max_distance 5
nodal_degree 0.244444
traffic_file 22_eu.trf
parallel_demands 160
link_cap 10
max_level 1
step_of_increase -1
stat_periods 500
method dpp
```

A 26 csomópontos észak-amerikai hálózat:

```
@nodes
label coords
0     (71,149)
1     (84,439)
2     (42,334)
3     (151,398)
4     (178,268)
5     (247,508)
6     (402,475)
7     (420,542)
8     (410,407)
9     (433,244)
10    (422,339)
11    (319,325)
12    (516,244)
13    (571,318)
14    (592,245)
15    (501,349)
16    (562,414)
17    (660,257)
18    (776,272)
19    (733,214)
20    (686,420)
21    (527,519)
22    (799,210)
23    (626,481)
24    (683,599)
```

```
25    (753,340)
@edges
              label cap   cost
0     2       0     1     856
0     4       1     1     866
1     2       2     1     427
1     3       3     1     298
1     5       4     1     881
2     0       5     1     856
2     1       6     1     427
2     4       7     1     753
3     1       8     1     298
3     4       9     1     476
3     5       10    1     733
4     0       11    1     866
4     2       12    1     753
4     3       13    1     476
4     11      14    1     480
5     1       15    1     881
5     3       16    1     733
5     6       17    1     707
5     7       18    1     840
6     5       19    1     707
6     7       20    1     275
6     8       21    1     301
6     11      22    1     822
6     16      23    1     783
7     5       24    1     840
7     6       25    1     275
7     21      26    1     403
8     6       27    1     301
8     10      28    1     284
8     15      29    1     451
9     10      30    1     509
9     12      31    1     438
10    8       32    1     284
10    9       33    1     509
10    11      34    1     681
10    15      35    1     299
11    4       36    1     480
11    6       37    1     822
11    10      38    1     681
12    9       39    1     438
12    13      40    1     230
12    14      41    1     317
12    15      42    1     327
13    12      43    1     230
13    15      44    1     289
13    16      45    1     310
13    17      46    1     345
14    12      47    1     317
14    17      48    1     119
15    8       49    1     451
15    10      50    1     299
15    12      51    1     327
15    13      52    1     289
16    6       53    1     783
16    13      54    1     310
```

```
16   20   55   1    415
16   23   56   1    286
17   13   57   1    345
17   14   58   1    119
17   19   59   1    520
17   25   60   1    386
18   19   61   1    184
18   22   62   1    234
18   25   63   2    269
19   17   64   1    520
19   18   65   1    184
19   22   66   1    182
20   16   67   1    415
20   23   68   1    286
20   25   69   1    416
21   7    70   1    403
21   23   71   1    532
21   24   72   1    826
22   18   73   1    234
22   19   74   1    182
23   16   75   1    286
23   20   76   1    286
23   21   77   1    532
23   24   78   1    749
24   21   79   1    826
24   23   80   1    749
25   17   81   1    386
25   18   82   2    269
25   20   83   1    416
@attributes
avg_distance 3.30769
max_distance 8
nodal_degree 0.309524
traffic_file 26_usa.trf
parallel_demands 78
link_cap 10
max_level 1
step_of_increase -1
stat_periods 500
method dpp
```

Kimenő file:

Kérésenkénti blokkolások száma:

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 3 4 4 5 6 6 6 7 8 8 8 8 8 8 9 9 10 10 11 11 12 13 14 15 15 16 16 17 17 17 17 17 18 18 19 19 20 21 22 23 24 25 26 27 28 29 29 30 31 32 33 33 34 35 36 36 37 38 39 40 41 42 43 44 45 46 46 46 47 48 48 48 49 50 51 52 52 53 53 54 55 56 57 57 58 58 59 60 61 61 61 61 61 62 63 64 65 65 66 67 67 68 69 70 71 72 73 74 74 75 76 77 78 79 79 80 81 81 82 82 83 84 85 86 87 88 89 89 90 91 92 93 94 95 96 97 97 98 99 100 101 102 102 103 104 105 106 106 107 108 109 110 111 112 113 113 113 114 115 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 138 138 139 140 141 142 142 143 143 144 145 146 147 148 149 150 151 152 152 153 154 155 156 157 158 159 160 161 162 163 164 165 165 166 167 167 168 169 170 171

172 172 172 173 174 175 176 177 178 179 179 180 181 182 183 184 185 186 187 188 189 190 191 191 192 193 194 195 196 197 198 199 200 200 201 202 203 204 205 206 207 208 209 210 211 212 212 213 214 215 216 217 218 219 220 221 222 222 223 224 225 226 227 228 229 230 230 231 232 233 234 235 236 237 238 238 239 239 240 241 242 243 243 244

SASP(200 kérés) futtatás után az európai hálózat élei:

0101001001011011110011100000000
0100000001011010110011100111110
0110000000000011110011100000110
0101001000011010000011011100000
0110111101000010000011011101110
0100111101011010110011100111110
0001101001011000111101110111100
0111100000110111100011100101110
0101101111011110101011100111100
0101010100110111101011101101110
0111101111011110111101110101110
0111000000110000000011100000000
0110000111011011100000000111100
0111101110011010000011101111100
0111001111011110100011110111110
0110010000000000111000001100000
0001000100110011110011101101110
0000111100000111101011100000110
0110110111011110110111011101110
0110100001110110111011001110100
0000010000011000000111011100000
0110001111010110111011100111110
0110001111010110110111011101110
0000110111011110111011111010000
0111011001011110101111011101110
0101110001011110101111000111110
0110111101110010111011101110100
0110111101010010110000000000000
0111101111011010110111011101110
0110110101000000111000000000110
0111100101011110110011110101110
0101100101010100000011100111100
0111101111011110111101110111100
0111011000110011110111000101110
0000001110011010110111011101110
0111010101010100000011100111100
0110011001011110000111100011110
0111100101010000000111010000000

01111001010111101011000111000
01110011100000001111000111011
01111011101100101101110111011
01111011110000100000111000001
01110100000000100000111000000
01111011110110101101110101100
00000011110111101101110100011

SASP-2 futtatás után a hálózat élei:

01000011010000000000111101001
01010111011010010010011100011
01100111001101110000000000010
01010100011010010010111101111
01100100001101110011000001111
01000011010000000011011101011
00011011010111001111011100000
01111011110111011010001101111
01011011110000001110100101111
01011011110111010110101101111
01111011110111011010111101111
01110000000111000010001100000
01110111000000110010000011000
01111000011010010010011100111
01110111101111001000000101111
01100000111000010001100001111
00000111001101110001101100001
01111011110111000110100001111
01110111101110110111011011011
01110111001110110111001101111
01110111101111000011100100011
01110111011011100111011010011
01001111011101001000010101111
01101110111011010111011101111
01101011000011000000000001111
01111011001110110111001101011
01100000111011010110010000011
01000111101111010110010101111
01011011000010001110010000101
01111011110111001111011111011
01110111011101110010001100011

00010111011011010110111101110
01110000000001001110010000101 0
01101111000001001011110000011 0
01000000011101000000010001111 0
01100111100001110000010101111 0
01110111000001110110111100000 0
01010111011101110110001101111 0
01000111011100110110000001111 0
01110000011011010000000000001 0
01111011011101001110110101111 0
00000111101110010000110001111 0