Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék

Sebők Tamás

MOHÓ ÚTVONALVÁLASZTÁS SZÁMÍTÁSIGÉNYÉNEK CSÖKKENTÉSE HEURISZTIKUS MÓDSZEREKKEL

KONZULENS

Dr. Gulyás András, Csernai Márton, Kőrösi Attila

BUDAPEST, 2013

Tartalomjegyzék

1 Bevezetés ... 3
2 A földrajzi útvonalválasztás ... 4
2.1 Definíció ... 4
2.2 Metrikus terek ... 4
2.3 Mohó továbbítás ... 5
2.4 Irány alapú továbbítás ... 7
3 A heurisztikák ... 10
3.1 Definíció ... 10
3.2 Használatuk okai ... 10
3.3 Sorrend alapú heurisztikák ... 10
3.3.1 Véletlen sorrend (RO) ... 12
3.3.2 Forgalom alapú sorrend (TO) ... 12
3.4 Irány alapú heurisztika – Előzetes számítások ... 13
3.4.1 Irány közelebb (ST) ... 14
4 A szimulációs környezet ... 16
4.1 A vizsgált metrikus terek ... 16
4.1.1 Távolságfüggvények ... 16
4.1.2 Szögszámító függvények ... 17
4.2 A vizsgált topológiák ... 17
4.3 A vizsgált tulajdonságok ... 19
5 A szimuláció eredménye ... 22
5.1 Az inicializáló keresések hatása a TO útvonalválasztásra ... 22
5.2 Az inicializálásokra fordított idő ... 22
5.3 A továbbító algoritmusok összehasonlítása ... 23
5.3.1 Sikeres útvonalkeresések aránya ... 24
5.3.2 Átlagos úthossz ... 25
5.3.3 Műveletek száma ... 25
5.3.4 Futási idő ... 26
6 Összefoglalás ... 28
Irodalomjegyzék ... 29

1 Bevezetés

A mohó továbbítás a földrajzi útvonalválasztás alapját képező módszer. A földrajzi útvonalválasztás gondolata először az 1980-as években merült fel rádiós hálózatokkal, és elkülönülő hálózatok összekapcsolásával [1] kapcsolatban. Jelenleg is főleg vezeték nélküli hálózatokban használják, a legelterjedtebb megoldás a face routingot is használó GPSR [2]. Van olyan megoldás is [3], ahol nem szükséges a földrajzi helyzet ismerete, ez főleg ad-hoc és szenzor hálózatok számára lett kidolgozva. A földrajzi útvonalválasztás azonban más területeken is használható, kutatások vannak például az alkalmazására közösségi, szociális hálók esetében is, sőt, vizsgálták a használatát Internet topológián is [4].

A mohó útvonalválasztás egy egyszerű ötletre épít: a hálózatban lokális döntések segítségével próbáljunk egy globálisan is jó útvonalat találni. A mohó útvonalválasztást nem csak az egyszerűsége (csak lokális információk kellenek a döntéshez) teszi népszerűvé, hanem az elosztott működése is. Nincs szükség a teljes hálózat ismeretére az útvonalválasztáshoz, nem kell az útvonalat előre meghatározni, az útvonal a továbbítás közben, lokális döntések következtében alakul ki.

Azonban a mohó továbbítással kapcsolatban vannak megoldandó problémák. A legnagyobb probléma a számításigénye, ugyanis a mohó algoritmus mindig a lokális optimumot keresi. Ehhez meg kell vizsgálni minden szomszédot, ezért meghatározása sok szomszéddal rendelkező csomópontok esetén költséges lehet. A dolgozatban azt mutatom meg, hogyha a továbbításnál lemondunk a lokális optimum megkereséséről, és heurisztika segítségével próbálunk egy lokálisan jó (de nem feltétlenül a legjobb) döntést hozni, akkor közel olyan jó megoldáshoz jutunk lényegesen kevesebb számítással, ezáltal sokkal gyorsabban.

2 A földrajzi útvonalválasztás

2.1 Definíció

A földrajzi útvonalválasztás onnan kapta a nevét, hogy az útvonalválasztás a földrajzi helyzeten alapul. Ennek megfelelően minden csomópontot a helyzetével (koordináták) azonosítunk, és a csomópontok a saját helyzetükön felül ismerik a szomszédjaik helyzetét is. Ha egy csomagot akarunk továbbítani, akkor ismernünk kell a cél címét is. Azonban a csomópontok ezen felül semmilyen egyéb információt nem tartanak fenn a hálózatról, nem kell ismerniük a hálózati topológiát, tehát nincs szükség a hálózat felderítésére és ilyen információk tárolására és megosztására. Ez jelentősen lecsökkenti a hálózat fenntartásához szükséges kommunikáció mennyiségét a hálózatokban, és az eszközök is kevesebb erőforrást igényelnek egy földrajzi útvonalválasztást használó hálózatban.

2.2 Metrikus terek

A földrajzi útvonalválasztásnál nincs megkötés arra, hogy a csomópontok helyzetét konkrétan hogyan írjuk le, tetszőleges metrikus térben használható. Metrikus térnek egy olyan halmazt nevezünk, amelyen értelmezve van egy távolságfüggvény (más néven metrika). A távolságfüggvény a halmazban lévő bármely elempárhoz egy nemnegatív valós számot rendel, amit az elemek távolságának nevezünk.

A távolságfüggvénytől az alábbi tulajdonságok teljesülését követeljük meg (x, y, z a halmaz tetszőleges elemei, d(x,y) a távolságfüggvény):

1. Két pont távolsága pontosan akkor 0, ha a két pont egybeesik.
   $d(x, y) = 0 \Leftrightarrow x = y$
2. A távolságfüggvény szimmetrikus.
   $d(x, y) = d(y, x)$
3. A távolságfüggvényre teljesülnie kell a háromszög egyenlőtlenségnek, azaz két pont távolsága nem lehet kisebb, ha egy köztes ponton keresztüli távolságot nézzük.
   $d(x, z) \leq d(x, y) + d(y, z)$

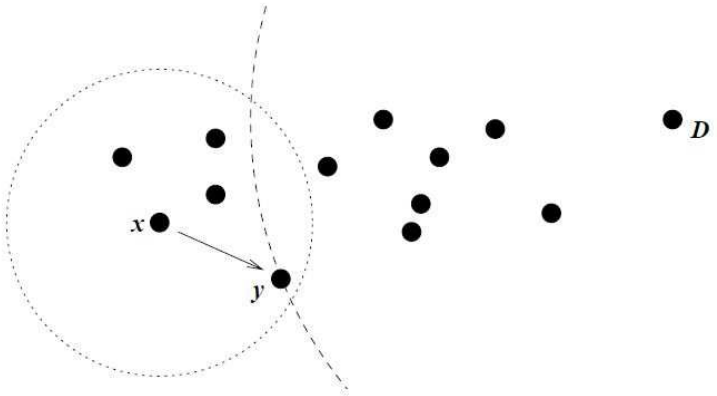
A legismertebb metrikus tér az Euklideszi-tér, ahol az Euklideszi távolságfüggvény szerint határozzuk meg a pontok távolságát. Az Euklideszi távolságfüggvény n dimenziós Euklideszi térben $d(x,y)=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$ alakban írható fel, ennek az általánosítása a Minkowski távolságfüggvény ($d(x,y)=\sqrt[p]{\sum_{i=1}^{n}|x_i-y_i|^p}$). A Minkowski távolságfüggvény p=1 esetben a Manhattan-távolságot adja, ami úgy fogható fel, mintha egy egységoldalú négyzethálónak a vonalain vett távolságot néznénk. A hiperbolikus és gömbi térben is megvannak a megfelelő távolságfüggvények, melyek a tér megfelelő függvényeit használják a távolság meghatározásához.

Vannak azonban más távolságfüggvények is, melyek tetszőleges halmazon teljesítik a három axiómát. Például a $d(x,y)=\begin{cases}0, ha\ x=y\\1, ha\ x\neq y\end{cases}$ függvény, ami minden nem egybe eső pont távolságát 1-nek veszi, vagy a $d(x,y)=\max_{1\le i\le n}\left\{|x_i-y_i|^2\right\}$ függvény. Ezekkel a függvényekkel csak azt szeretném megmutatni, hogy a metrika használata nem jelent megszorítást, azonban a metrikák eltérő tulajdonságai miatt fontos a megfelelő metrika kiválasztása.

2.3 Mohó továbbítás

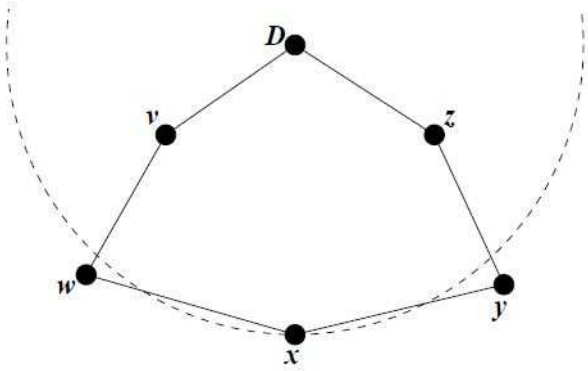
A földrajzi útvonalválasztásnál a routing során használhatjuk a mohó algoritmust a továbbításra. A mohó továbbítás teljesen elosztott módon működik, az útvonal minden pontjában lokális információk alapján történik a döntés, hogy merre történjen a továbbítás. A mohó továbbítás lényege, hogy az adott pontban elérhető összes alternatívát megvizsgálva annak a csomópontnak történik a továbbítás, ami az adott pontban a legjobb választás. A hagyományos mohó továbbításnál a csomópont kiszámítja az összes szomszédjának a céltól vett távolságát (a megfelelő metrikus térben), és annak a szomszédjának továbbítja a csomagot, amelyik a célhoz legközelebb esik.

A mohó továbbítás menetét szemlélteti az 1. ábra, ahol x pont felől akarunk D csomópontnak továbbítani egy csomagot. A továbbításhoz x pont az y szomszédját használja, mivel az van a legközelebb a cél csomóponthoz, azaz D-hez.



1. ábra A mohó továbbítás [2]

Ahhoz, hogy a mohó továbbítás ne kerüljön végtelen ciklusba, leállási feltételt kell megadnunk. A leállási feltétel lehet például az, hogy csak addig történjen továbbítás, amíg a célhoz közelebb kerülünk. Ilyenkor azonban mohó továbbítás nem minden esetben sikeres, ezt mutatja a 2. ábra. Az ábrán látszik, hogy x mindkét szomszédja (y és w) is távolabb van a céltól, D-től mint x, ezért a továbbítás leáll, hiába lenne út w-n vagy y-on keresztül is D-be.



2. ábra A mohó továbbítás nem minden esetben sikeres [2]

A mohó továbbítás sikeressé tehető például az ú.n. „face” routing segítségével, így garantálni lehet a sikeres továbbítást, amennyiben létezik útvonal a forrás és a cél csomópont között. A face routing technológia azt az esetet tudja orvosolni, amikor a mohó algoritmus lokális optimumba kerül, és segítségével a routing ilyenkor folytatható [5].

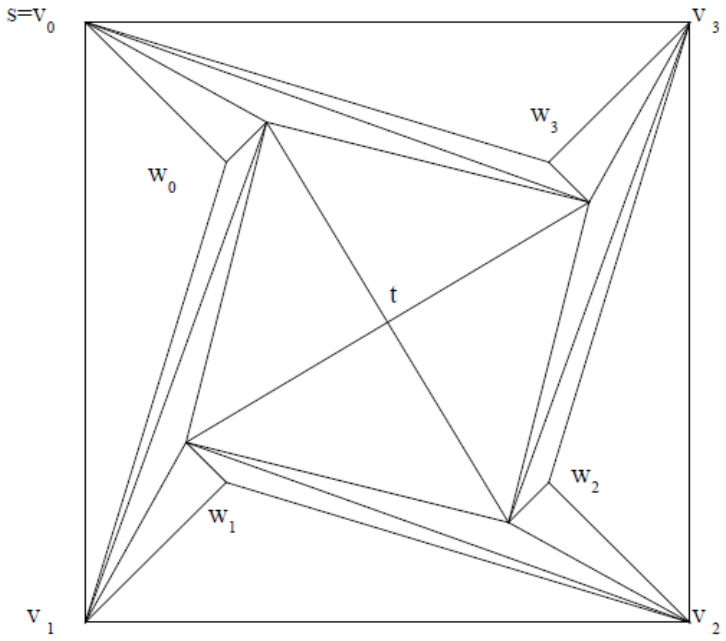
Az alábbiakban látható a mohó algoritmus pszeudokódja. A pszeudokódban megadott algoritmus akkor áll le, ha nem tud közelebb kerülni a célhoz. A szimulációkban ennek a pszeudokódnak az implementációját használom.

```
Adott G(V,E) gráf; egy forrás és egy cél csomópont
begin
    aktuális = forrás
    előző = Ø
    while(aktuális != cél és előző != aktuális)
          akt_táv = távolság(aktuális, cél)
          legközelebbi = aktuális
          for szomszed sz in szomszédjai(aktuális)
                szomsz_táv = távolság(sz, cél)
                if (szomsz_táv < akt_táv)
                      akt_táv = szomsz_táv
                      legközelebbi = sz
                end if
          end for
          előző = aktuális
          aktuális = legközelebbi
    end while
end
```

2.4 Irány alapú továbbítás

A földrajzi útvonalválasztásnál a mohó továbbítás mellett más módszerek is rendelkezésre állnak a továbbításra. A mohó továbbításhoz hasonlóan működik az irány alapú továbbítás, csak itt az algoritmus nem a legközelebbi, hanem a céllal legjobban egyező irányba eső szomszéd felé továbbít.

Azonban a továbbítás során végtelen ciklusba kerülhetünk, ezt mutatja a 3. ábra. Akármelyik v_i pontból indulunk ki, a legjobban egyező irányú szomszédnak továbbítás során soha nem fogunk eljutni t pontba, ugyanis v_i-ből w_i esik a legközelebb t irányához, ahonnan azonban v_{i+1}-be vezet az algoritmus, mivel annak az iránya hasonlít a legjobban t irányához. Pedig az ábrán látszik, hogy 2 lépés alatt bármelyik v_i-ből eljuthatnánk t-be. A végtelen ciklus elkerülésének érdekében be kell vezetni egy lépésszám (hopcount) határt, aminek elérésekor az algoritmus leáll.



3. ábra Az irány alapú továbbítás végtelen ciklusba kerülhet [6]

Az algoritmus pszeudo kódja az alábbiakban látható. Fontos eltérés a mohó továbbításhoz képest, hogy itt ismernünk kell a hálózat tulajdonságait, és ezen tulajdonságok alapján kell beállítani a lépésszám korlátot. Túl alacsony korlát esetén csökkenhet a sikeresség, mivel a távolabbi csúcsok közti hosszabb útvonalakat kiszűri a korlát. Túl magas korlát esetén viszont a hatékonyság romolhat olyan végtelen ciklusba került továbbításokkal, amik sose lesznek sikeresek.

```
Adott G(V,E) gráf; egy forrás és egy cél csomópont; és egy lépészám_korlát
begin
    aktuális = forrás
    lépésszám = 0
    while(aktuális != cél és lépésszám < lépészám_korlát)
          továbbítás_iránya = irány(aktuális, cél)
          különbség = ∞
          for szomszéd sz in szomszédjai(aktuális)
                szomszéd_iránya = irány(sz, cél)
                if (eltérés(továbbítás_iránya, szomszéd_iránya) < különbség)
                      továbbítás_iránya = szomszéd_iránya
                      következő = sz
                end if
          end for
          aktuális = következő
    end while
end
```

Az algoritmus sikerességét lehet javítani a face routing technológia segítségével (lásd Compass Routing II, [6]), akárcsak a mohó továbbítást.

3 A heurisztikák

3.1 Definíció

A heurisztika lassan, vagy pontosan nem megoldható problémák megoldására használt technika, mely bizonyos engedmények (optimalitás, teljesség, pontosság terén) mellett szolgáltat közelítő megoldásokat.

A földrajzi útvonalválasztásnál azért használok heurisztikát, hogy gyorsabban, kevesebb művelet segítségével, ezáltal erőforrás-takarékosabban találjak egy, az optimális útvonalhoz hasonló útvonalat. Ez különösen fontos nagyméretű, egyszerűbb eszközöket tartalmazó hálózatoknál. Ilyenek például a szenzorhálózatok, ahol az eszköz energia fogyasztását döntően határozza meg az adatok továbbítására használt energia. A heurisztika által talált útvonal nem feltétlenül optimális, így szükséges annak vizsgálata, hogy mennyire tér el az optimális útvonaltól.

3.2 Használatuk okai

A földrajzi útvonalválasztás egyik legnagyobb előnye, hogy nem eltárolt routing táblák alapján továbbít, ezáltal nincs szükség routing táblák nyilvántartására, megosztására és frissítésére. Cserébe minden továbbításkor meg kell hozni egy döntést a cél címe alapján, hogy merre történjen a továbbítás, ezért fontos, hogy ennek a döntésnek a meghozása minél egyszerűbben és gyorsabban megtörténjen.

Mohó továbbításkor a döntés meghozása annál tovább tart, minél több opciót kell az algoritmusnak figyelembe vennie. Az opció továbbítás esetén a szomszédokat jelenti, azaz minél több szomszéddal rendelkezik egy csomópont, annál több továbbítási lehetőséget kell vizsgálnia. Ez azért különösen nagy probléma, mert jellemzően minél több szomszéddal rendelkezik egy adott csomópont a hálózaton, annál nagyobb forgalom megy át rajta, tehát annál többször kell futtatnia a mohó továbbító algoritmust.

3.3 Sorrend alapú heurisztikák

Amikor elkezdtem azon gondolkodni, hogy lehetne a mohó továbbítást gyorsítani, az első gondolatom az volt, hogy lehetőleg minél kevesebb szomszéd vizsgálatára legyen szükség továbbításkor. Ez rögtön két kérdést is felvet: Ha nem

vizsgálom az összes szomszédot, akkor mi alapján fogadok el egy szomszédot a továbbítás szempontjából jónak? Milyen sorrendbe vizsgálom a szomszédokat?

Ezen heurisztikáknál enyhítek az optimalitás feltételén, tehát továbbításkor nem követelem meg, hogy az optimális, a célhoz legközelebbi szomszédnak történjen a továbbítás, megelégszem azzal is, ha a továbbítással a célhoz közelebb jutunk. Tehát a továbbítás feltétele sorrend alapú heurisztikáknál, hogy közelebb kerüljünk a cél csomóponthoz. Fontos megjegyezni, hogy a mohó továbbítás leállási feltétele megmaradt, tehát csak addig történik továbbítás, amíg a célhoz közelebb jutunk.

A sorrend alapú heurisztikák működésének lényege, hogy előre meghatározott sorrendben ellenőrzik az adott csomópont szomszédjait, és amint találnak egy olyan szomszédot, ami kielégíti a továbbítás feltételét (közelebb visz a célhoz), megtörténik a továbbítás. Minden csomóponthoz tartozik egy sorrend, ami a csomópont minden szomszédját pontosan egyszer tartalmazza, így minden szomszédot legfeljebb egyszer vizsgál továbbításkor a sorrend alapú heurisztika. A sorrend alapú heurisztikák működését mutatja az alábbi pszeudokód. A szomszéd_sorrend függvény adja vissza a csomópont szomszédjainak rendezett listáját.

```
Adott G(V,E) gráf; egy forrás és egy cél csomópont
begin
    aktuális = forrás
    előző = Ø
    while(aktuális != cél és előző != aktuális)
          akt_táv = távolság(aktuális, cél)
          lista = szomszéd_sorrend(aktuális)
          legközelebbi = aktuális
          while(lista != {Ø} és legközelebbi==aktuális)
                n = következő_elem(lista)
                szomsz_táv = távolság(n, cél)
                if (szomsz_táv < akt_táv)
                      legközelebbi = n
                end if
          end while
          előző = aktuális
          aktuális = legközelebbi
    end while
end
```

Mivel a gyengébb továbbítási feltételt legalább annyi, de jellemzően több csomópont is kielégíti, mint a mohó továbbításnál, és a mohó továbbításnál minden szomszéd ellenőrzése megtörténik, itt pedig csak addig vizsgáljuk a szomszédokat, amíg egy közelebbit nem találunk, ezért legfeljebb annyi, de jellemzően sokkal

kevesebb szomszéd vizsgálata is elég. Az, hogy pontosan mennyi szomszéd vizsgálata szükséges egy csomópontban, az erősen függ a felállított sorrendtől.

A sorrend alapú heurisztikák eltérő útvonalakat adhatnak, és emiatt a továbbítás sikeressége is eltérő lehet a hagyományos mohó továbbításhoz képest, azonban ha egy csomópontban a hagyományos továbbítás sikeres, akkor a sorrend alapú heurisztika alapján történő továbbítás is sikeres lesz.

A továbbiakban megadok két sorrend alapú heurisztikát, amit implementáltam, azonban más sorrend alapú heurisztikák (például fokszám alapú heurisztika) is elképzelhetőek. A két megadott heurisztika a véletlen sorrend (RO, Random Order), és a forgalom alapú sorrend (TO, Traffic Order).

3.3.1 Véletlen sorrend (RO)

Véletlen (random) sorrend alapú heurisztikánál egy adott csomópontban a csomag továbbításakor a csomópont szomszédjait véletlen sorrendben vizsgálom. Nagy előnye az egyszerűsége, ugyanis a többi sorrend alapú heurisztikával ellentétben nem igényel bonyolult, előzetes sorrend felállítást, azonban a véletlen sorrendnél sokkal jobb sorrendek is vannak, ahol hamarabb találni egy megfelelő szomszédot.

Fontos megjegyezni, hogy nem véletlenszerűen sorsolja az adott csomópont szomszédjait vizsgálatra, hanem egy listán megy végig, ami véletlen sorrendben tartalmazza a szomszédokat. Bár nem tűnik lényegesnek az eltérés, mégis nagyon lényeges, mivel így lehet garantálni, hogy az algoritmus minden szomszédot maximum egyszer vizsgál, és egy adott csomópontban legfeljebb annyi vizsgálat történik, mint a hagyományos mohó továbbításnál.

3.3.2 Forgalom alapú sorrend (TO)

A forgalom alapú sorrendnél arra a gondolatra építek, hogyha felmérjük az éleken átmenő forgalmat, és a csomópont szomszédjait az oda vezető él forgalma alapján sorrendezzük (nagyobb forgalmú előrébb szerepel), akkor jobb eredményt kapunk a véletlen sorrendnél. Így a hálózaton előzetesen hagyományos mohó továbbítást használva kereséseket (csomagtovábbításokat) végzek, melynek során rögzítem az éleken átmenő forgalmat. Fontos, hogy az előzetes keresés során ugyanolyan, véletlen kereséseket hajtok végre, mint amit majd a szimulációnál is tesztelni fogok, hogy jobban közelítsem a szimulációban is előforduló forgalmat. A

forgalom felmérése után minden csomópontnál létrehozzuk a szomszédok sorrendjét tartalmazó listát.

Az előzetes, inicializáló keresések száma fontos paramétere a forgalom alapú sorrendnek, így érdemes vizsgálni, hogy ezt a paramétert változtatva mi történik.

3.4 Irány alapú heurisztika – Előzetes számítások

Az irány alapú heurisztika bizonyos szempontból hasonlít a sorrend alapú heurisztikákra, azonban egy teljesen eltérő ötletre épül. A hasonlóság alapja, hogy az irány alapú heurisztikánál is minden csomópontnak van egy rendezett listája a szomszédjairól.

Már a forgalom alapú heurisztika kitalálásakor is láttam, hogy nehéz egy kellően jó sorrend kialakítása. Ugyanis elvégezhetünk sok mohó keresést a hálózatban, amivel meg tudjuk állapítani, hogy általában merre kell továbbítani (és aztán ezt felhasználhatjuk vizsgálati sorrendként), azonban ez nem veszi figyelembe a konkrét esetet, hogy a keresés honnan jött, és merre tart.

Arra jutottam, hogy legyen egy olyan lista, ami irány szerint rendezve tartalmazza a csomópont szomszédjait. Kell egy referencia irány (az origó iránya), és ehhez az irányhoz képest nézzük meg a csomópont szomszédjaiba vezető élek irányát, tehát hogy az él a referencia iránnyal milyen szöget zár be. Ez a szög értelemszerűen 0 és 2π (radiánban kifejezve, szögben kifejezve 0^o és 360^o) közt adódik. A szögek ismeretében már létre lehet hozni a csomópont szomszédjainak irány szerint rendezett listáját. Fontos, hogy olyan struktúrát kell létrehozni, amiben később az irány alapján tudunk keresni, és megtudni, hogy az adott irányba milyen szomszéd található.

Az irány alapú heurisztikánál ezután arra használjuk fel a rendezett listát, hogy továbbításnál gyorsan ki tudjunk választani egy adott irányba eső szomszédot. Ugyanis az irány alapú továbbítás a mohó továbbítással ellentétben meggyorsítható úgy, hogy a szomszédokat előre rendezzük egy referencia irányhoz képest, így elég a cél és a referencia irány közti különbséget kiszámolni és megkeresni az adott irányhoz legközelebb eső szomszédot a rendezett listában. A csomópontok irány szerint rendezett szomszédlistájának előállítását mutatja az alábbi pszeudokód:

```
Adott G(V,E) gráf
begin
    for all v in V
          szomszéd_lista={}
          for szomszéd sz in szomszédjai(v)
                irány = irány_számítás(origo, v, sz)
                irány = irány_korrekció(v, sz, irány)
                beszúrás(szomszéd_lista, sz, irány)
          end for
    end for
end
```

A pszeudokódban az irány_számítás függvény végzi el a szögszámítást az origó, az aktuális csomópont, és a szomszédos csomópont közt. Ez a szögszámítás a koszinusz tételre vezethető vissza (ismert a háromszög három oldala, és az egyik csúcsánál lévő szögre vagyunk kíváncsiak), azonban a különböző metrikus terekben különbözőképpen kell számolni. Miután a kapott szög 0 és π közötti, a szomszédok listája pedig 0 és 2π közti, így korrigálni kell a csúcs és a szomszédjának helyzete alapján, ezt végzi az irány_korrekció függvény. A beszúrás függvény ezután beszúrja a csúcs szomszédjainak listájába a megfelelő helyre (ezt az angle paraméter határozza meg) az adott szomszédot.

3.4.1 Irány közelebb (ST)

Ennél a heurisztikánál a legfontosabb módosítás a hagyományos irány alapú továbbításhoz képest, hogy minden továbbítás előtt megvizsgálja a csomópont, hogy az adott szomszédnak továbbítás közelebb visz-e a célhoz. Amennyiben nem, eltér az irány alapú továbbítástól, és próbál egy jobb döntést hozni. Ez esetben a szomszédok rendezett listájában a szomszédtól jobbra illetve balra eső szomszédok közül kiválasztja azt, amelyik irányban közelebb esik a cél irányához, és megvizsgálja, hogy az közelebb visz-e. Egészen addig lépked ebben a listában, amíg nem talál egy közelebbi szomszédot, vagy körbe nem ér. Ha nincs olyan szomszéd, aki közelebb vinne, az algoritmus leáll.

Nagy előnye ennek a heurisztikának, hogy a fenti leírt működés segítségével el lehet kerülni az irány alapú továbbításnál látott végtelen ciklusba kerülést, nem szükséges megadni lépésszám korlátot, és egy olyan hálózaton, ahol a mohó keresés mindig sikeres, ez a heurisztika is mindig sikeres lesz. A heurisztika használatához

viszont olyan adatstruktúrában kell tárolni a szomszédlistát, ami mindkét irányba járható (például kétirányú láncolt lista, tömb).

4 A szimulációs környezet

4.1 A vizsgált metrikus terek

A vizsgálataim a kétdimenziós euklideszi, gömbi és hiperbolikus terekre terjedtek ki, így az ezek használatához szükséges távolság és szögszámító függvényeket implementáltam. A program indulásakor meg kell adni, hogy milyen modellt (és ez által milyen metrikát) használjon. A következőkben megadom az implementálásra került távolság és szögszámító függvényeket. A szögszámító függvények jellemzően az adott térre vonatkozó koszinusz tételből jönnek ki.

4.1.1 Távolságfüggvények

- Euklideszi tér:
  - Ha a pontok x, y koordinátákkal rendelkeznek:
    $a = (x_1, y_1),\ b = (x_2, y_2)$
    $d_{euclidean}(a,b) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
    Mivel a gyökfüggvény a pozitív számokon (négyzetszámok összege sosem negatív) monoton nő, ezért ha a cél csak a legközelebbi szomszéd megtalálása, akkor elhagyható a gyökvonás (kisebb szám négyzetgyöke kisebb)
  - Polár koordináta rendszer (a pontok sugárral (r) és szöggel (γ) vannak azonosítva):
    $a = (r_1, \gamma_1),\ \mathrm{b} = (\mathrm{r}_2, \gamma_2)$
    $x_1 = r_1 * \cos(\gamma_1),\ \mathrm{y}_1 = r_1 * \sin(\gamma_1)$
    $x_2 = r_2 * \cos(\gamma_2),\ \mathrm{y}_2 = r_2 * \sin(\gamma_2)$
    A koordináták átalakítása után a távolságszámítás a $d_{euclidean}$ képlet alapján történik.
- Hiperbolikus tér:
  - Poincaré disk modell [7] (a pontok x, y koordinátákkal rendelkeznek):

$$a = (x_1, y_1),\ b = (x_2, y_2)$$

$$d_{poincaré}(a,b) = \cosh^{-1}\left(1 + \left(2 * \frac{d_{euclidean}(a,b)}{(1 - d_{euclidean}(0,a)) * (1 - d_{euclidean}(0,b))}\right)\right)$$

- Hiperboloid modell [7] (a pontok sugárral (r) és szöggel (γ) vannak azonosítva):

$$a = (r_1, \gamma_1),\ \mathrm{b} = (\mathrm{r}_2, \gamma_2)$$

$$d_{hyperboloid}(a,b) = \cosh^{-1}(\cosh(r_1) * \cosh(r_2) - \sinh(r_1) * \sinh(r_2) * \cos(\gamma_1 - \gamma_2))$$

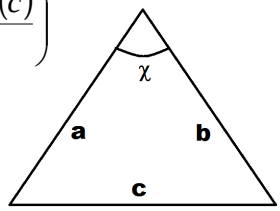
- Gömbi tér:
  - Ha a pontok sugárral (r) és szöggel (γ) vannak azonosítva, a gömb sugara pedig ρ ($\rho = 1$ alapértelmezésben):

$$a = (r_1, \gamma_1),\ \mathrm{b} = (\mathrm{r}_2, \gamma_2)$$

$$d_{spherical}(a,b) = \rho * \cos^{-1}\left(\cos\left(\frac{r_1}{\rho}\right) * \cos\left(\frac{r_2}{\rho}\right) + \sin\left(\frac{r_1}{\rho}\right) * \sin\left(\frac{r_2}{\rho}\right) * \cos(\gamma_1 - \gamma_2)\right)$$

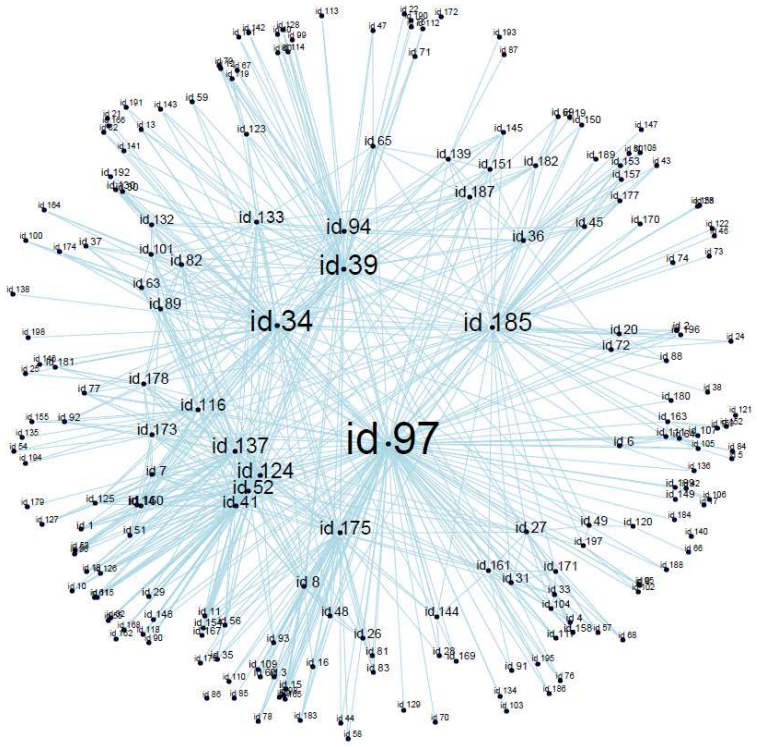
4.1.2 Szögszámító függvények

- Euklideszi tér: $\chi = \cos^{-1}\left(\frac{a^2 + b^2 - c^2}{2ab}\right)$
- Hiperbolikus tér: $\chi = \cos^{-1}\left(\frac{\cosh(a) * \cosh(b) - \cosh(c)}{\sinh(a) * \sinh(b)}\right)$
- Gömbi tér: $\chi = \cos^{-1}\left(\frac{\cos(c) - \cos(a) * \cos(b)}{\sin(a) * \sin(b)}\right)$



4.2 A vizsgált topológiák

Korábbi vizsgálataim során azt az eredményt kaptam, hogy a hiperbolikus terek használata előnyös a földrajzi útvonalválasztás és a topológiák építése szempontjából is, sok valós hálózat mögött is hasonló természetű terek húzódnak. E miatt a továbbító algoritmusok működését egy kisméretű (200 csomópontból álló), hiperbolikus térbe ágyazott hálózaton teszteltem.



4. ábra A továbbító algoritmusok teszteléséhez használt topológia

Egy ilyen méretű hálózaton még jól lehet követni a továbbító algoritmusok egyes lépésit, de már kellően nagyméretű ahhoz is, hogy a továbbító algoritmusok közti eltéréséket fel lehessen deríteni. Ezt a hálózatot mutatja a 4. ábra, a hálózatra a továbbiakban hyperbolic próbaként hivatkozok.

A továbbító algoritmusok működését kipróbáltam egy nagyobb (több mint 23000 csomópontot tartalmazó) hiperbolikus térbe ágyazott Internet topológiával is. Ez a topolólógia egy tudományos cikk [4] eredményeként állt elő, ahol azt vizsgálták, hogy az Internetet alkotó AS-ek (Autonomous System, azaz autonóm rendszer) hogyan kapcsolódnak egymáshoz, majd a kapott hálózatot egy speciális eljárással hiperbolikus térbe ágyazták. Ez a hálózat is nagyon jó alapot nyújt a továbbító algoritmusok vizsgálatához, erre a topológiára a továbbiakban hyperbolic AS topológiaként hivatkozok.

Továbbá kipróbáltam az algoritmust speciálisan generált hálózatokon is. Egy saját TOP_GEN algoritmussal paramétereiben valós hálózatokhoz hasonló, greedy routolható (ahol a mohó keresés minden esetben sikeres) topológiákat hoztam létre. Az alábbi pszeudokód mutatja a generáláshoz használt algoritmust:

```
Adott G(V,E) gráf, ahol E={}
for all v in V
    fed_távok={}
    for all u in V\v
        fed_távok=fed_távok ∪ min_k(d(k,v)|d(k,u)<d(v,u))
    end for
    max_táv=max(fed_távok)
    for all u in V\v
        if d(u,v)<max_táv
            E=E ∪ (u,v)
        end if
    end for
end for
```

Ezen algoritmus segítségével tetszőleges méretű hálózatok generálhatóak, teszteléshez 200 csomópontú hálózatokat, mérésekhez 10000 csomópontból álló euklideszi, hiperbolikus, és gömbi térbe elhelyezett hálózatokat generáltam. Ezen hálózatokon a mohó továbbítás minden esetben sikeres, így kiválóan alkalmasak a különböző heurisztikák kipróbálására és összehasonlítására.

4.3 A vizsgált tulajdonságok

Ahhoz, hogy a továbbító algoritmusokat össze tudjuk vetni egymással, meg kell határozni, hogy az útvonalválasztás mely tulajdonságait vizsgáljuk. Ezeket a tulajdonságokat definiálom a következőkben.

Sikeres útvonalkeresések aránya

A szimulációnál véletlenszerűen sorsolt pontpárok között végzek útvonalkereséseket. A sikeres útvonalkeresések arányát úgy kapjuk meg, hogy a sikeres útvonalkeresések (amikor talált az adott továbbító algoritmussal utat a két pont között) számát elosztjuk az összes útvonalkeresés számával (sikeresség = sikeres keresések száma / keresések száma). Az útvonalkeresések végrehajtása előtt a program ellenőrzi, hogy az adott két pont között létezik-e útvonal, és ha nincs útvonal, akkor nem hajt végre útvonalkeresést, így a sikeresség arányát nem rontják azok az esetek, amikor nem is létezik út a két pont között.

Átlagos úthossz

Az útvonalkeresések során kapott útvonalhosszak átlaga. Az útvonal hossza alatt a megtett lépések számát értem (nem pedig a metrikus térben megtett út hosszát). A továbbító algoritmusok összevethetőségének érdekében csak azok az útvonalkeresések

számítanak bele az átlagba, amikor az összes vizsgált továbbító algoritmus sikeres útvonalkeresést eredményezett.

Elvégzett műveletek száma

Ahhoz, hogy össze lehessen vetni a továbbító algoritmusok komplexitását, bevezettem egy absztrakt művelet (Operation, „O") fogalmat. Mivel a ténylegesen elvégzett műveletek száma architektúra és implementációfüggő, emiatt nehezen meghatározható, így magasabb szintű műveleteket definiáltam. Ezek a hagyományos mohó továbbításnál a következőek:

Vizsgálat, hogy a cél a szomszédjaim közt van-e ($O_{sz}(x)$)

Ez a művelet akkor hajtódik végre, ha a beérkező csomag (vagy keresés) címzettje nem az aktuális csomópont. Ha az aktuális csomópont közvetlen szomszédja a cél csomópont, akkor ez az egyetlen művelet történik, és az aktuális csomópont továbbítja a csomagot a cél csomópontnak (ami a közvetlen szomszédja).

Távolságszámítás ($O_{tsz}(x,y)$)

Két csomópont (x, y) közti távolság számítása. Ehhez szükség van a két csomópont koordinátáira, valamint arra, hogy milyen metrikát és távolságszámítást használunk (ezt nevezem modellnek, a modell a szimuláció előtt rögzítésre kerül). Távolságszámításra akkor kerül sor, hogyha a csomag címzettje nem az aktuális csomópont, és nem is valamelyik szomszédja. Ilyenkor az aktuális csomópont először a saját, majd később a szomszédjai célhoz képesti távolságát számítja ki.

Távolságok összehasonlítása ($O_{th}(t_1,t_2)$)

Két távolság (t_1, t_2) összehasonlítása. Ha az aktuális csomóponthoz képest a szomszédos csomópontok mind távolabb vannak a céltól, akkor mohó továbbítás sikertelen. Ha van az aktuális csomópontnál közelebbi szomszéd, akkor a legközelebbi felé történik a továbbítás.

Egy csomag mohó továbbítása során az útvonal egy x pontjában elvégzett műveletek száma:

$$M(x) = \begin{cases} O_{sz}(x), \text{ha a cél x szomszédja} \\ O_{sz}(x) + O_{tsz}(x, cél) + \sum_{i=1}^{N} \left(O_{tsz}(n_x(i), cél) + O_{th}\left(t_{n_x(i)}, t_{\min}\right)\right), \text{egyébként} \end{cases}$$

N: x szomszédjainak száma;

$n_x(i)$: x i-edik szomszédja;

t_{min}: az eddigi nézett csomópontok közül a legközelebbi távolsága (kezdetben x távolsága)

A sorrend alapú heurisztikáknál is ugyanezek a műveletek hajtódnak végre, az eltérés a műveletek számában adódik. Ugyanis a sorrend alapú heurisztikáknál a szomszédokat a rendezett lista alapján veszi sorba, és nem veszi minden esetben az összes (N darab) szomszédot, csak addig megy, ameddig egy közelebbit nem talál.

Az irány alapú továbbítás műveletszámának meghatározásához azonban új műveletek definiálása szükséges, ezek a műveletek a következőek:

Szög számítása ($O_{szög}(x,y)$)

Szögszámítást az irány és a tartomány meghatározásakor használok. A szögszámítás egy koszinusz tételre vezethető vissza, egy háromszög három oldalának ismeretében könnyen meghatározható egy adott csúcsnál lévő szöge a szögfüggvények segítségével. A háromszöget a két csúcs (x,y), és az origó adja.

Szomszéd választása ($O_{val}(d)$)

A csomópont szomszédjainak irány szerint rendezett listájából egy szomszéd választása. Ennél a műveletnél előjön a csomópontok közti különbség, mivel egy rendezett listából egy elem kivételéhez szükséges lépések száma az elemek számával logaritmikusan arányos, tehát ennek a műveletnek a konkrét lépésszáma függ attól, hogy az adott csomópontnak hány szomszédja van.

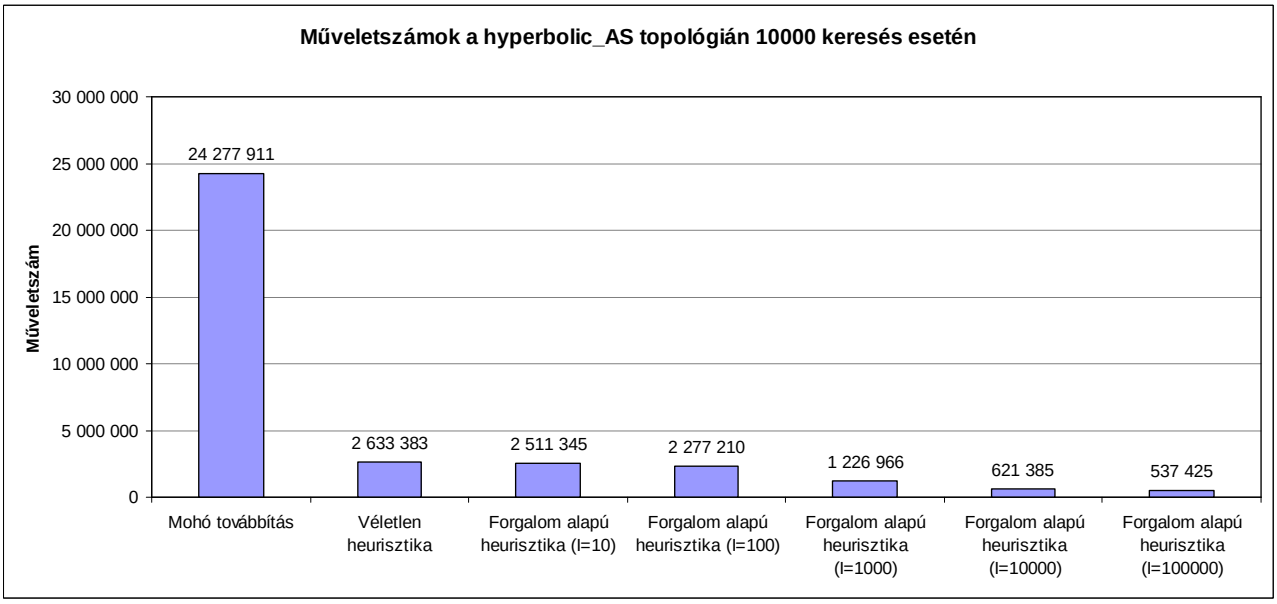
Az útvonalkeresések által igényelt processzoridő (futási idő)

Ugyan az útvonalkeresések által használt processzoridő mérése erősen függ az útvonalkeresések, és az onnan hívott függvények implementálásától, azonban ez az adat áll a legközelebb a valósághoz, ez mutatja meg legjobban a továbbító algoritmusok számításigénye közti különbséget (nem olyan absztrakt, mint az előbb definiált műveletek). A szimulációban így minden továbbító algoritmussal történő útvonalkeresés futtatása előtt és után lekérdezem a processzortól az aktuális időt (ezt mikroszekundum pontossággal adja vissza a rendszer), és amennyiben az összes továbbító algoritmussal sikeres útvonalkeresés történt (erre az összehasonlíthatóság miatt van szükség), az eltelt időt rögzítem.

5 A szimuláció eredménye

5.1 Az inicializáló keresések hatása a TO útvonalválasztásra

A forgalom alapú sorrendet használó heurisztikánál rendkívül fontos, hogy hány inicializáló keresést hajtunk végre, mielőtt a heurisztikát élesben használnánk. Az 5. ábrán látható, hogy a véletlen heurisztika nagyjából tizedére csökkenti a műveletek számát a mohó továbbításhoz képest valós hiperbolikus térbe ágyazott AS topológia esetében. Amennyiben az inicializáló keresések száma (ezt I-vel jelölöm) elhanyagolható (I=10), akkor a forgalom alapú heurisztika a véletlen heurisztikához hasonló eredményt ad. Amennyiben növeljük az inicializáló keresések számát, a forgalom alapú heurisztika annál jobb eredményt ad, a műveletszám akár a mohó továbbításnál tapasztalt 2%-ára csökkenthető, ez látható az 5. ábrán.



5. ábra Műveletszámok az AS hálózaton 10000 keresés esetén (sorrend alapú heurisztikák)

5.2 Az inicializálásokra fordított idő

Ugyan csak egyszer kell elvégezni, de nem elhanyagolható az az időráfordítás, amit az inicializálásra szánunk. Az 1. táblázat azt mutatja, hogy a szimuláció során mennyi időbe telt a különböző továbbításokhoz használt inicializálások végrehajtása (a teljes idő, az összes csomópontra, a hiperbolikus AS gráfon). Látszik, hogy irány alapú továbbításnál nincs jelentős eltérés aközt, hogy egy vagy két irányban bejárható listát

készítünk a csomópontok szomszédjairól. A forgalom alapú heurisztikánál látható, hogy 100 inicializáló keresés esetén nagyjából ugyanannyi az inicializálásra fordított idő, mint az irány alapú heurisztikák esetén. Az 5. ábra alapján azt kapjuk, hogy legalább 1000 inicializáló keresésre van szükség ahhoz, hogy a forgalom alapú heurisztika a véletlen heurisztikánál lényegesen kevesebb műveletet végezzen, az 1. táblázatból pedig az is kiderül, hogy ehhez nagyjából ötször annyi inicializálási időre van szüksége, mint az irány alapú továbbításnak.

Inicializálási idők Hyperbolic_AS gráfon	
Inicializálás	Szükséges idő (micro sec)
Irány alapú sorrend	136 612
Irány alapú sorrend (mindkét irányba bejárható)	137 920
Forgalom alapú heurisztika (l=10)	105 477
Forgalom alapú heurisztika (l=100)	162 708
Forgalom alapú heurisztika (l=1000)	806 964
Forgalom alapú heurisztika (l=10000)	6 995 527
Forgalom alapú heurisztika (l=100000)	70 513 435

1. Táblázat A továbbító algoritmusok inicializálására fordított idők a Hyperbolic AS gráfon

5.3 A továbbító algoritmusok összehasonlítása

A 2. táblázat mutatja a kapott eredményeket négy különböző topológiára. A négy topológia három 10000 csomópontú, különböző (hiperbolikus, gömbi /szférikus/, euklideszi) térben TOP_GEN algoritmus segítségével generált, valamint az AS hálózatból áll.

hyp 10000 csp gráf, 100000 keresés	Mohó	Sorrend alapú heurisztika		Irány alapú heurisztika	Irány (listával)
		RO	TO (l=1000)	ST	(limit=100)
Inicializálási idő (micro sec)	0	0	307 892	97 339	93 215
Futási idő (micro sec)	50 654 902	10 457 775	5 651 883	2 342 847	1 798 776
Műveletek száma	107 742 817	20 501 751	20 302 445	1 407 337	1 084 705
Sikeres keresések aránya	1	1	1	1	0.9523
Átlagos úthossz	3.43	6.96	6.37	3.57	3.6

sph 10000 csp gráf, 100000 keresés	Mohó	Sorrend alapú heurisztika		Irány alapú heurisztika	Irány (listával)
		RO	TO (l=1000)	ST	(limit=100)
Inicializálási idő (micro sec)	0	0	128 633	51 052	49 827
Futási idő (micro sec)	24 442 377	16 442 166	6 033 149	6 736 971	6 622 539
Műveletek száma	69 969 068	41 009 856	27 229 502	22 754 112	18 220 716
Sikeres keresések aránya	1	1	1	1	0.9938
Átlagos úthossz	40.4	63.38	49.68	46.59	46.59

euc 10000 csp gráf, 100000 keresés	Mohó	Sorrend alapú heurisztika		Irány alapú heurisztika	Irány (listával)
		RO	TO (l=1000)	ST	(limit=100)
Inicializálási idő (micro sec)	0	0	138 766	43 912	42 815
Futási idő (micro sec)	25 951 258	16 927 911	6 215 898	6 494 038	6 283 191
Műveletek száma	71 317 511	41 948 589	28 093 437	23 135 023	18 524 811
Sikeres keresések aránya	1	1	1	1	0.9982
Átlagos úthossz	41.02	64.35	50.85	47.15	47.15

AS gráf, 100000 keresés	Mohó	Sorrend alapú heurisztika		Irány alapú heurisztika	Irány (listával)
		RO	TO (l=1000)	ST	(limit=100)
Inicializálási idő (micro sec)	0	0	820 105	137 920	136 612
Futási idő (micro sec)	89 589 018	13 758 426	4 181 385	3 381 647	2 548 166
Műveletek száma	201 339 043	25 468 061	13 151 107	1 776 944	798 081
Sikeres keresések aránya	0.8198	0.8951	0.9008	0.8343	0.7894
Átlagos úthossz	3.67	5.71	5.48	4.05	4.03

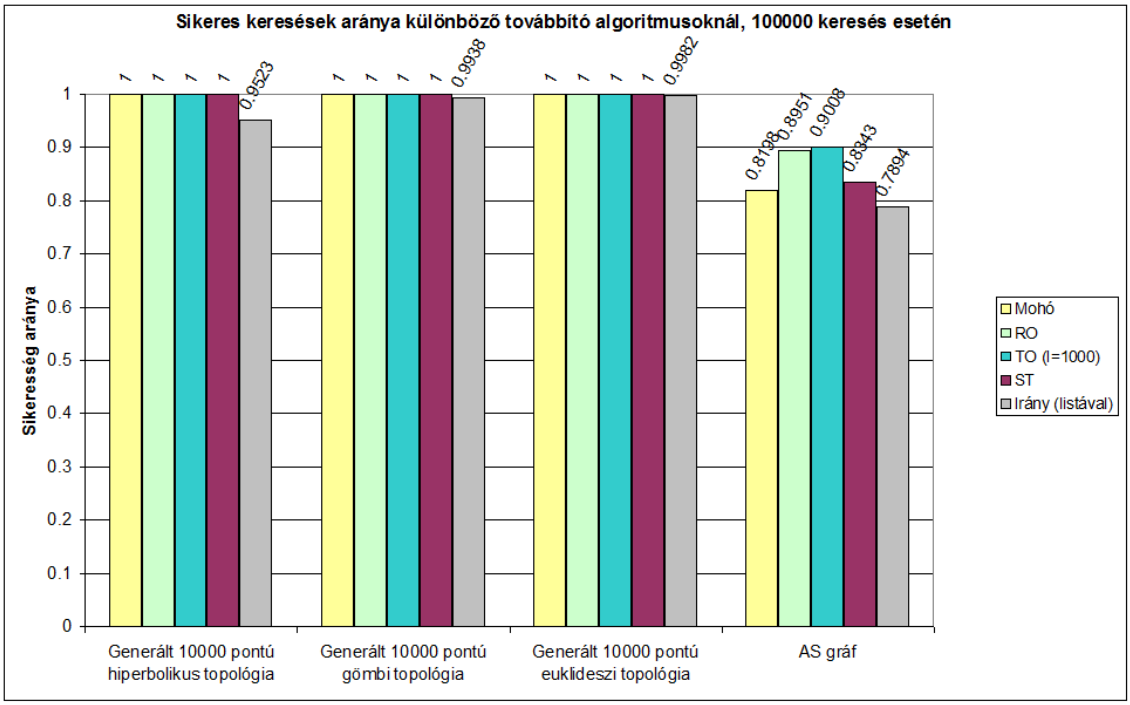
2. Táblázat A továbbító algoritmusok eredményei különböző topológiákon

A táblázat alapján összevethetőek a különböző terekbe ágyazott azonos méretű hálózatok tulajdonságai is, azonban én a továbbiakban azt vizsgáltam, hogy egy adott topológián a különböző továbbítási módszerek hogyan viselkednek.

5.3.1 Sikeres útvonalkeresések aránya

A generált hálózatokon az irány alapú továbbítást (ez szerepel a táblázatban irány (listával) néven, jelezve, hogy itt használva van az előre elkészített szomszéd lista) leszámítva minden továbbító algoritmus minden esetben sikeresen működött, az irány alapú az esetek 0.1-5%-ban volt sikertelen. Ennek oka, hogy egy greedy routeolható hálózaton nem feltétlen sikeres az irány alapú továbbítás minden esetben, emiatt kell az ST heurisztika, ugyanis az minden esetben sikeres.

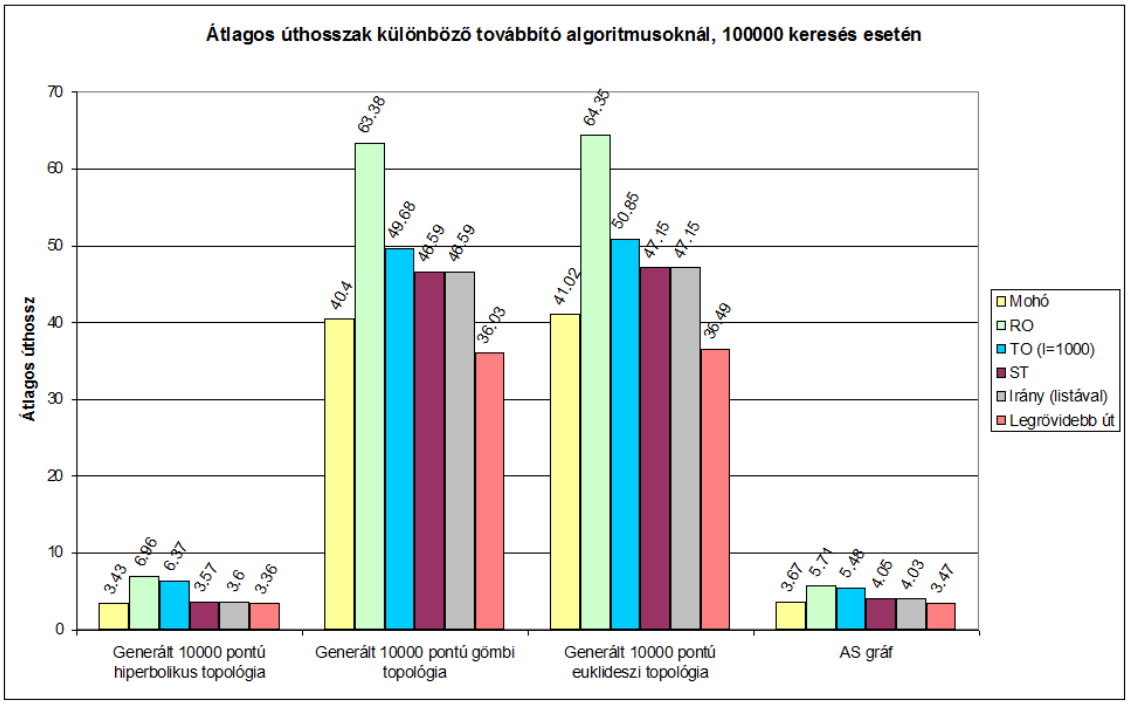
Az AS topológián már a hagyományos mohó továbbítás se volt minden esetben sikeres, ezen a heurisztikák pont amiatt tudtak javítani, hogy gyengébb továbbítási feltételük nagyobb bolyongást enged meg a hálózatban, ezáltal jobban el tudják kerülni a lokális minimumokat. A vártnak megfelelően ezen a topológián is az irány alapú továbbítás szerepelt a legrosszabbul. Az eredményeket grafikus formában a 6. ábra mutatja.



6. ábra A sikeres keresések aránya

5.3.2 Átlagos úthossz

Átlagos úthossznál azt kapjuk, hogy a mohó továbbítás úthossza közelíti meg legjobban a létező legrövidebb (shortest) út hosszát. A legnagyobb a véletlen heurisztika útvonalnyúlása, ezután következik a forgalom alapú (7. ábra). Fontos látni, hogy az irány alapú továbbítás és heurisztika közelíti meg a legjobban a mohó továbbítás úthosszát. Azaz az irány alapú továbbítással és heurisztikával lehet a mohó továbbításnál kapott útvonalakhoz hasonlóan rövid utat találni. E kettő között azért nem látható lényeges eltérés az adatoknál, mert a táblázat csak azon keresésekre vonatkozó adatokat mutat, ahol minden továbbító algoritmus sikeres volt, és sikeres irány alapú továbbítás esetén csak ritka esetekben (ha irány alapján választott szomszéd nem visz közelebb a célhoz, de később az irány alapú továbbítás mégis sikeres lesz) van eltérés a két algoritmus által adott útvonal között.



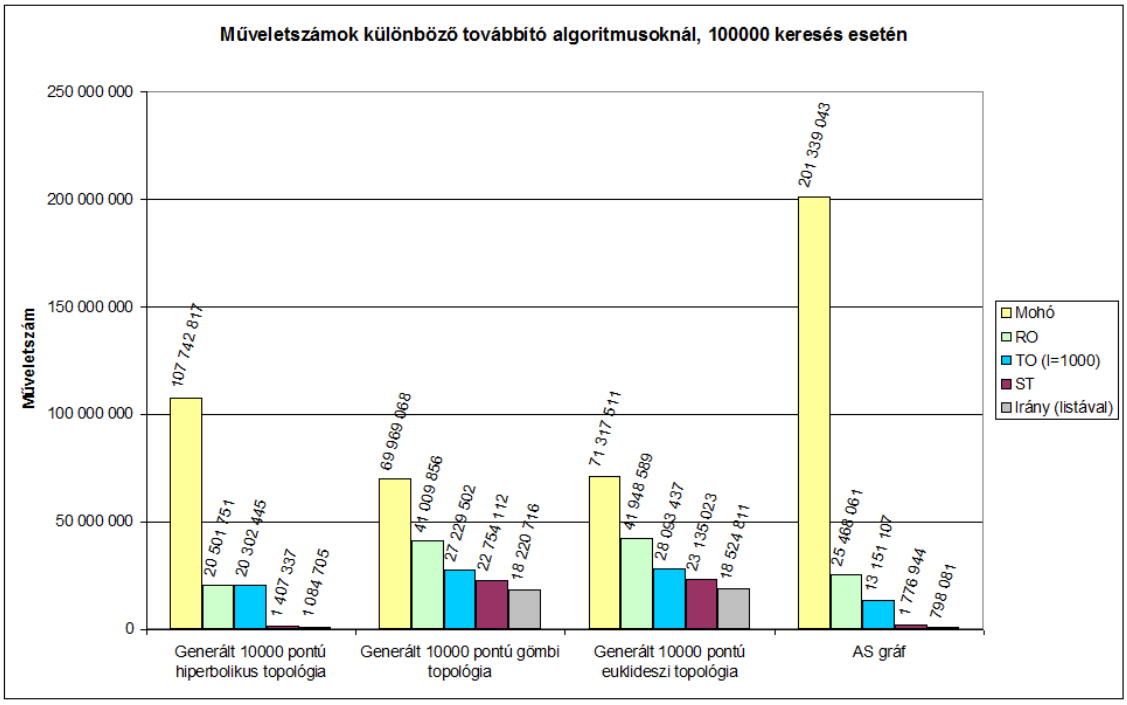
7. ábra Átlagos úthosszak

5.3.3 Műveletek száma

A műveletek számánál (8. ábra) jól látható a továbbító algoritmusok komplexitása közti különbség. A mohó továbbítás műveletigénye a legnagyobb, az AS gráfon például kilencszerese a rangsorban utána lévő véletlen heurisztikának, és kétszázötvenszerese a műveletszám szempontjából legjobb irány alapú továbbításnak. Ezt a hálózatban lévő nagy fokszámú csúcsok okozzák, amiken rendkívül sok keresés

megy keresztül. De a sorrend minden topológián megegyezik: a mohót a véletlen sorrend (RO) követi, majd a forgalom alapú sorrend (TO) jön, ezután az irány alapú továbbítások.

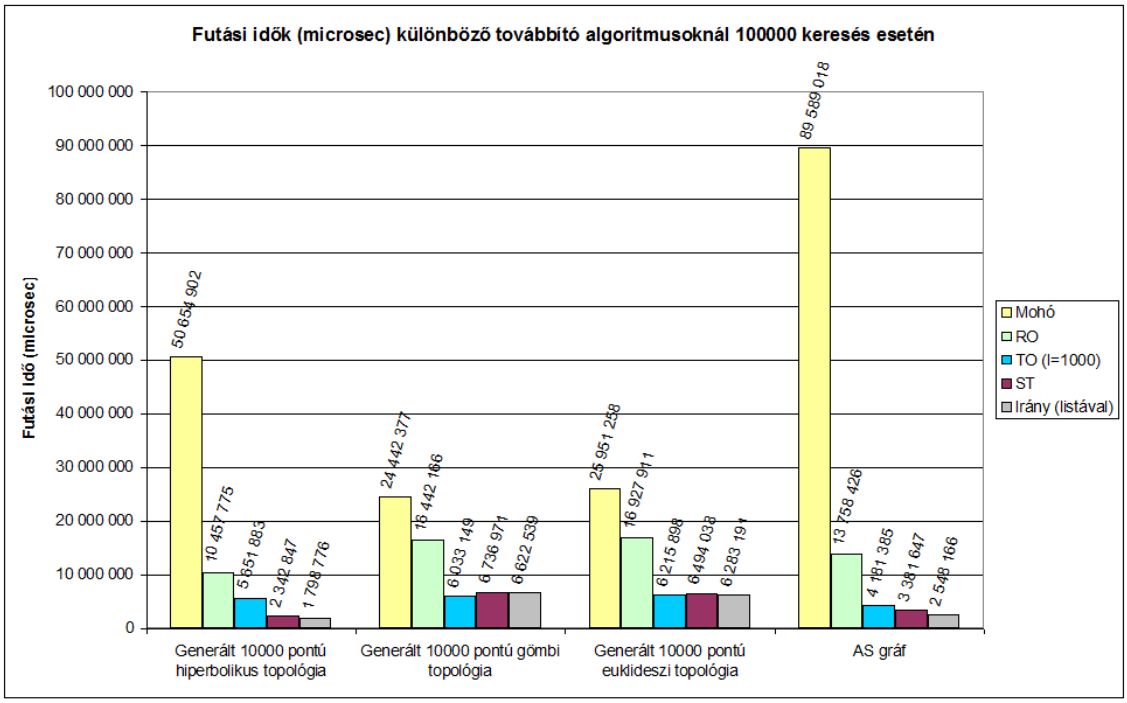
Jól látszik, hogy az ST ahhoz, hogy garantálja a továbbítás sikerességét, mennyi plusz műveletet végez az irány alapú továbbításhoz képest. Fontos megjegyezni, hogy a mérésekben szereplő irány alapú továbbítás az ST-hez hasonlóan használja az előre letárolt szomszéd listát, és az adatok itt is az irány alapú továbbításnak kedveznek, mivel az összehasonlítás nem mutatja a sikertelen továbbításoknál végzett műveletek számát, amikor látszódna a bonyolultabb továbbító algoritmusok előnye.



8. ábra Műveletszámok

5.3.4 Futási idő

Futási idő szempontjából hasonló mondható el, mint a műveletek számánál, az ábrák (8. és 9.) is hasonlítanak, de a különbsége árnyaltabbak.



9. ábra Futási idők

A sorrend változatlan, a mohó továbbítás futásideje messze a legmagasabb. A második helyen a véletlen heurisztika (TO) áll minden topológián. A futási idő szempontjából legjobban teljesítő három továbbító algoritmusnál (RO, ST, irány) viszont nincsenek jelentős különbségek, hasonlóan jól teljesítenek minden topológián. Ez egyben azt is jelenti, hogy annak az ellenőrzése az ST továbbításnál, hogy közelebb kerülne-e a célhoz, a futási idő szempontjából nem jelent jelentős többletet a (szomszéd listát használó, azaz gyorsított) irány alapú továbbításhoz képest.

6 Összefoglalás

A tervezett továbbítási algoritmusok eltérő tulajdonságokkal rendelkeznek, így nem lehet röviden összefoglalni az eredményeket. Jól látszik az eredményekből, hogy egy irány szerint rendezett szomszédlistával nagyon gyorssá lehet tenni az irány alapú továbbításokat. Ez a gyorsaság még akkor is megmarad, ha a végtelen ciklusok elkerülésére és a továbbítás sikerességének növelése érdekében az ST továbbítást használjuk, ami ellenőrzi, hogy a továbbítással közelebb kerülnénk-e a célhoz. A forgalom alapú heurisztika is sok szempontból hasonlóan jó eredményeket tud felmutatni mint az ST, ehhez képest az ST továbbítás legnagyobb előnye, hogy kevesebb az inicializálásra fordított idő. Az minden heurisztikára igaz, hogy a mohó továbbításhoz képest rendkívüli mértékben meg tudja gyorsítani a továbbítást a számításigény csökkentésével.

Irodalomjegyzék

[1] Finn, Gregory G. : *Routing and Addressing Problems in Large Metropolitan-Scale Internetworks*, University of Southern California, 1987 március
http://www.isi.edu/div7/people/finn.home/routing_and_addressing_problems_in_large_metropolitan-scale_internetworks.BW.pdf

[2] B. Karp and H. Kung: *Greedy Perimeter Stateless Routing*. In Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000), 2000.
http://www.eecs.harvard.edu/~htk/publication/2000-mobi-karp-kung.pdf

[3] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, I. Stoica. *Geographic Routing without Location Information*. In Proceedings of the 9th annual international conference on Mobile computing and networking (Mobicom 2003), 2003.
http://www.cs.ucla.edu/classes/fall03/cs218/paper/p96-rao.pdf

[4] M. Boguñá, F. Papadopoulos, and D. Krioukov: Sustaining the Internet with Hyperbolic Mapping, Nature Communications, v.1, 62, 2010
http://www.nature.com/ncomms/journal/v1/n6/full/ncomms1063.html#/abstract
http://arxiv.org/pdf/1009.0267.pdf

[5] Prosenjit Bose, Pat Morin, Ivan Stojmenović, Jorge Urrutia: *Routing with Guaranteed Delivery in Ad Hoc Wireless Networks*, Wireless Networks, November 2001, Volume 7, Issue 6, pp 609-616
http://cg.scs.carleton.ca/~morin/publications/online/unitgraphs-wn.pdf

[6] Evangelos Kranakis, Harvinder Singh, Jorge Urrutia: *Compass Routing on Geometric Networks*, IN PROC. 11 Th Canadian Conference on Computational Geometry, 1999
http://www.matem.unam.mx/~urrutia/online_papers/comprout.pdf

[7] Bővebb leírás található a Poincaré Disk Modellről és a Hyperboloid modellről, és az ezekben használt távolságfüggvényről a Wikipédia angol nyelvű oldalain:
http://en.wikipedia.org/wiki/Poincar%C3%A9_disk_model
http://en.wikipedia.org/wiki/Hyperboloid_model