Algoritmusok viselkedése skálafüggetlen hálózatokban

Szakdolgozat

Készítette: Kiss Péter
Matematika BSc.
alkalmazott matematikus szakirány

Témavezető: Király Zoltán
egyetemi docens
Számítógéptudományi Tanszék

Eötvös Loránd Tudományegyetem
Természettudományi Kar
2022

Tartalomjegyzék

Köszönetnyilvánítás

Elsősorban témavezetőmnek, Király Zoltánnak szeretnék köszönetet mondani. Hálás vagyok hasznos tanácsaiért, ötleteiért és segítségéért, amikor szükségem volt rá. A szakdolgozatom elkészítése során felmerült kérdéseimre adott részletes válaszai nélkül ezen dolgozat nem jöhetett volna létre.

Köszönettel tartozom még a szüleimnek és testvéremnek, hogy minden helyzetben mellettem álltak, türelemmel és megértéssel támogattak.

1. Bevezetés

A szakdolgozatom témája a gráfalgoritmusok vizsgálata skálafüggetlen hálózatokban, amik számos tulajdonság szempontjából jól modellezik a való életben fellelhető hálózatokat, mint például a szociális hálózatokat, az idegsejtek alkotta hálózatot vagy az Internetet. Megvizsgálom, hogy az általános gráfokon legjobbnak vélt algoritmusok ezen speciális gráfokon is jobb teljesítményt érnek-e el, mint a lassabbnak hitt alternatíváik. Egyes algoritmusokon módosításokat hajtok végre, hogy jobban kihasználhassák a valós hálozatokon megfigyelt olyan főbb tulajdonságokat, mint a hatványfüggvényt követő fokszámeloszlást, a kisvilág-tulajdonságot és a magas klaszterezettséget.

Mivel véletlen gráfokat is vizsgálok, ezért egyes eredmények csak bizonyos valószínűséggel teljesülnek. Ezen állításokban a tulajdonságok általában $(1 - \mathcal{O}(n^{-1}))$ valószínűséggel (másnéven nagy valószínűséggel) vagy $(1-o(1))$ valószínűséggel (másnéven aszimptotikusan majdnem biztosan) teljesülnek.

A skálafüggetlen hálózatok legfontosabb tulajdonságainak definiálása után bemutatom ezen hálózatok generálására leggyakrabban használt modelleket: a Barabási-Albert modellt és a hiperbolikus véletlen gráfok modelljét. A dolgozatom első felében a maximális folyam keresés problémakört vizsgálom Thomas Bläsius, Tobias Friedrich és Christopher Weyand: Efficiently computing maximum flows in scale-free networks munkájuk alapján [1]. A Dinic-algoritmus módosított változatát hasonlítom össze a módosítatlan változat, a Boykov-Kolmogorov-algoritmus, és a szintező algoritmus teljesítményével valódi és generált hálózatokon.

A 4. fejezetben a minimális lefogó csúcshalmaz keresés problémakörével foglalkozom. A feladat definiálása után rámutatok a probléma nehézségére általános gráfok esetén, és bemutatom a 2 legismertebb közelítő algoritmust. Ezután Thomas Bläsius, Tobias Friedrich és Maximilian Katzmann: Efficiently approximating vertex cover on scale-free networks with underlying hyperbolic geometry munkájuk alapján a hiperbolikus véletlen gráfokon bemutatok egy $\mathcal{O}(m \log(n))$ idejű algoritmust, ami aszimptotikusan majdnem biztosan $(1 + o(1))$ approximációs hányadossal rendelkezik [8].

2. Skálafüggetlen hálózatok

2.1. Definíció. *Egy hálózatot skálafüggetlennek nevezünk, ha a fokszámeloszlása hatványfüggvényt követ:*

$$P(k) \sim k^{-\gamma}$$

ahol γ értékére gyakran teljesül a $2 < \gamma < 3$ egyenlőtlenség.

Ezen tulajdonság megnyilvánulása, hogy a gráfjaink néhány nagyon nagy fokszámú csúccsal (melyeket hub-oknak is szoktak nevezni) és sok alacsony fokszámú csúccsal fognak rendelkezni.

2.2. Definíció. *Egy hálózatot kisvilág-tulajdonságúnak nevezünk, ha a csúcsok közötti átlagos távolság nagyságrendje legfeljebb a csúcsok számának logaritmusa.*

Ezt a jelenséget először Karinthy Frigyes említette 1929-ben a Láncszemek című novellájában:

> „Tessék egy akármilyen meghatározható egyént kijelölni a Föld másfél milliárd lakója közül, bármelyik pontján a Földnek – ő fogadást ajánl, hogy legföljebb öt más egyénen keresztül, kik közül az egyik neki személyes ismerőse, kapcsolatot tud létesíteni az illetővel, csupa közvetlen – ismeretség – alapon".

Az elnevezés Stanley Milgram kisvilág-kísérletéből (1967) származik, aki felmérte, hogy 2 véletlenszerűen választott amerikai között átlagosan 5 személyes ismeretségi kapcsolaton keresztül lehet eljutni egy embertől egy másikig.

A klaszterezettség vagy klaszterezettségi együttható a gráfelméletben azt mutatja meg, hogy egy gráfban a csúcsok szomszédai által feszített részgráfok mennyire állnak közel a teljes gráfhoz.

2.3. Definíció. *Egy irányítatlan gráf csúcsának a klaszterezettségi együtthatója azt mutatja meg, hogy mennyire gyakori, hogy a csúcs szomszédai egymásnak is szomszédai.*
Az i-edik csúcs klaszterezettségi együtthatója:

$$C_i = \frac{2 \cdot |\{e_{jk} : v_j, v_k \in N_i, e_{jk} \in E\}|}{k_i \cdot (k_i - 1)} \tag{1}$$

ahol N_i az i. csúcs szomszédainak halmazát, és k_i ezen szomszédok számát adja meg. A teljes gráf klaszterezettsége az egyes csúcsok klaszterezettségi együtthatóinak az átlaga.

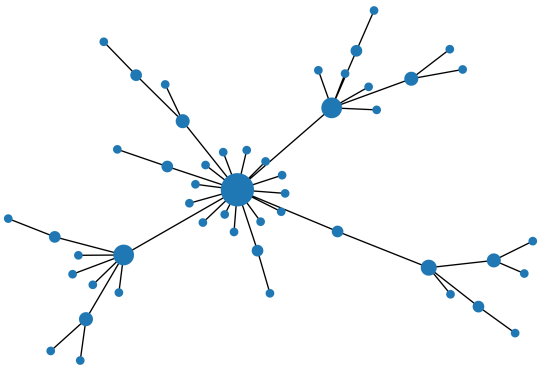
2.1. Barabási–Albert modell

A Barabási-Albert modell egy rekurzív algoritmus skálafüggetlen hálózatok generálására. A modellt Barabási-Albert László és Albert Réka definiálta 1999-ben [4].

Az algoritmusnak 3 paramétere van: $m_0 \geq 2$ csúcsú kezdeti hálózat, n a felépítendő gráfunkban a csúcsok száma, és m amennyi régi csúcshoz kapcsolunk egy újonnan érkezett csúcsot. Kiindulunk az m_0 gráfból és minden egyes lépésben hozzáadunk egy újabb csúcsot a gráfhoz mindaddig, amíg n csúcsunk nem lesz. Az új csúcsot az úgy nevezett preferenciális kapcsolódás módszerével adjuk hozzá a hálózathoz, azaz m darab véletlenszerűen választott régi csúcshoz kötjük egy-egy éllel úgy, hogy a kiválasztás valószínűsége arányos a régi csúcsok pillanatnyi fokszámával. Legyen p_i az a valószínűség, hogy az új csúcsunk az i-edik csúcshoz kapcsolódik, ekkor:

$$p_i = \frac{k_i}{\sum_{j=1}^{n} k_j}$$

ahol k_i az i-edik csúcs fokszáma, n az eddigi csúcsok száma. Ha az n jóval nagyobb m_0-nál, akkor ezzel a módszerrel kapott irányítatlan összefüggő gráf csúcsainak fokszámeloszlása fordítottan arányos lesz a fokszám köbével. Az 1. ábrán látható egy példa egy n=50, m=1 paraméterű Barabási-Albert modellből származó gráfra.



1. ábra. Barabási-Albert modellből származó gráf $m = 1$ esetén

2.2. Hiperbolikus véletlen gráfok

A Krioukov és társai által bemutatott hiperbolikus véletlen gráfok modelljéről megmutatták, hogy a kapott gráf fokszámeloszlása hatványfüggvényt követ, klaszterezettségi értéke magas és átmérője kicsi [2]. A modell által kapott gráfokra hiperbolikus geometriai gráfok néven is szoktak hivatkozni, mivel a geometriai gráfok osztályába sorolhatók, és elnevezéstől függően HRG-nek vagy HGG-nek rövidítik. A modell pontos bemutásához először is a hiperbolikus sík néhány tulajdonságának ismertetése szükséges.

A hiperbolikus sík pontjait modellezhetjük egy O középpontú r sugarú nyílt körlap pontjaival, ahol a körlapon kijelölünk egy úgynevezett referencia sugarat. Egy p pontot a 2 polárkoordinátája alapján tudjuk azonosítani: az első koordináta az origótól való távolságot jelenti, amit $r(p)$-el jelölünk, a második koordináta a $\varphi(p)$ szög, ami a p-n és O-n átmenő egyenes és a referencia sugár által bezárt szöget jelenti. A távolságot két tetszőleges p és q csúcs között a következőképpen definiáljuk:

$$\mathrm{dist}(p,q) = \mathrm{acosh}\left(\cosh(r(p))\cosh(r(q)) - \sinh(r(p))\sinh(r(q))\cos(\Delta_\varphi(p,q))\right)$$

ahol a $\Delta_\varphi(p,q) = \pi - |\pi - |\varphi(p) - \varphi(q)||$, ami a 2 helyvektor közötti legkisebb szöget jelenti. A hiperbolikus síkban egy r sugarú kör területe $2\pi(\cosh(r) - 1)$, és a kerülete $2\pi\sinh(r)$, azaz a sugár hosszát növelve mindkét mennyiség exponenciálisan növekszik.

A hiperbolikus véletlen gráfoknak 3 paramétere van: n a csúcsok száma, k az átlagos fokszám várható értéke, és egy α paraméter, ami a fokszámeloszlás hatványkitevőjét befolyásolja. Az n csúcsú hiperbolikus véletlen gráfokat úgy kapjuk meg, hogy n pontot egymástól függetlenül választunk az R sugarú nyílt körlapon belül és pontosan akkor kötjük őket össze, ha a hiperbolikus távolságuk legfejlebb R. A sugár hosszát úgy válasszuk meg, hogy $R = 2\log\frac{n}{k}$ teljesüljön. Egy csúcs választásánál a szög koordinátát egyenletes eloszlás szerint választjuk a $[0, 2\pi)$ intervallumból és az $r \in [0, R]$ koordinátát, azaz az origótól való távolságot az alábbi eloszlás szerint választjuk:

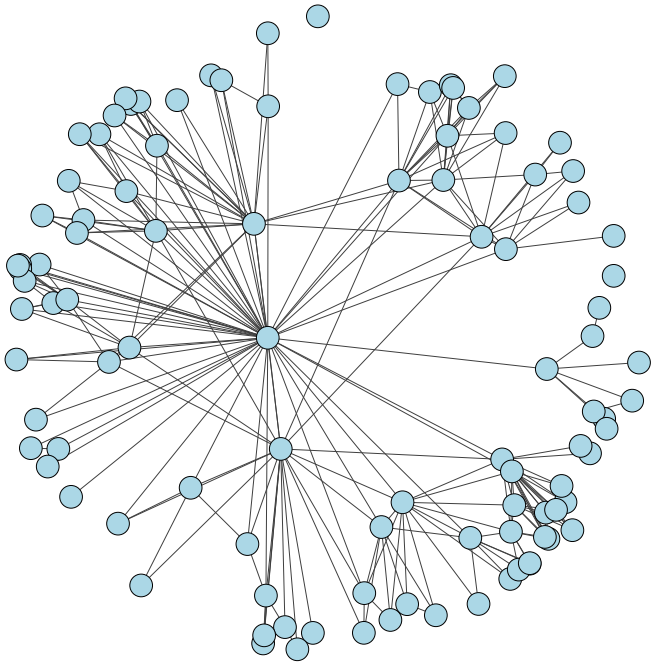
$$p(r) = \alpha\frac{\sinh(\alpha r)}{\cosh(\alpha R) - 1}$$

ahol $\alpha \in (\frac{1}{2}, 1)$ a gráfmodell úgynevezett növekedési paramétere. Alacsony α értékek esetén a csúcsok az origóhoz közelebb sűrűsödnek, míg nagyobb értékek esetén a körvonalhoz közelebb sűrűsödnek. Tehát a 2 polárkoordináta együttes sűrűségfüggvénye $r \in [0, R]$ és $\varphi \in [0, 2\pi)$ esetén:

$$f(r,\varphi) = \frac{1}{2\pi}\frac{\alpha\sinh(\alpha r)}{\cosh(\alpha R) - 1} = \frac{\alpha}{2\pi}e^{-\alpha(R-r)}(1 + \Theta(e^{-\alpha R} - e^{-2\alpha r})) \qquad (2)$$

Megmutatható, hogy az így kapott gráf fokszámeloszlása hatványfüggvényt követ, aminek kitevője $\beta = 2\alpha + 1$. Továbbá belátható, hogy a gráf egy óriási $\Omega(n)$ méretű komponenssel rendelkezik és a további komponensek legfeljebb polilogaritmikus méretűek a 2.1. tétel értelmében [3]. A 2.2. ábrán egy $n = 100$, $k = 1$, $\alpha = 0.6$ paraméterű hiperbolikus véletlen gráfra látható példa.

2.1. Tétel. *Legyen $2 < \beta < 3$. A hiperbolikus véletlen gráf második legnagyobb komponense legfeljebb $\mathcal{O}((\log n)^{\frac{2}{3-\beta}})$ méretű $1 - \mathcal{O}(\log n^{3/2})$ valószínűséggel.*



2. ábra. Hiperbolikus véletlen gráf

3. Maximális folyam keresés

A maximális folyam keresés probléma a legalapvetőbb gráfproblémák közé tartozik, és rendszeresen részfeladatként jelenik meg a gyakorlati alkalmazásokban. Például használják a vízvezetékeknél és a közlekedési hálózatoknál vagy a klaszterezési feladatoknál is [5]. A gyakorlatban legtöbbször a szintező algoritmust, a Boykov-Kolmogorov-algoritmust (röviden BK-algoritmust) és a Dinic-algoritmust használják maximális folyam keresésére. Közülük általános gráfon a szintező algoritmus teljesít legjobban, amit a legmagasabb aktív csúcs választással szoktak használni.

A legfontosabb definíciók után röviden bemutatom a fent említett algoritmusokat, és a Dinic-algoritmust módosítom egy olyan algoritmussá, ami várhatóan felülmúlja a versenytársai teljesítményét skálafüggetlen hálózatokon.

3.1. Definíció. *Legyen $G = (V, E)$ irányított gráf, $c\colon E \to \mathbb{R}^+$, $s \neq t \in V$. Ekkor a (G, c, s, t) rendszert hálózatnak, az s csúcsot forrásnak, a t csúcsot nyelőnek, és a c függvényt kapacitás függvénynek nevezzük.*

3.2. Definíció. *Az $f\colon E \to \mathbb{R}$ függvényt folyamnak nevezzük, ha az alábbiak teljesülnek:*

- *$\forall e \in E$-re $0 \leq f(e) \leq c(e)$*
- *$\forall v \in V \setminus \{s, t\}$-re $\sum_{u:uv\in E} f(uv) = \sum_{w:vw\in E} f(vw)$.*

A $\sum_{v:sv\in E} f(sv)$ mennyiséget folyamértéknek nevezzük és $|f|$-tel jelöljük. Egy uv élet telítettnek hívunk, ha $f(u, v) = c(u, v)$, különben pedig telítetlennek nevezzük.

3.3. Definíció. *Legyen $(G = (V, E), c, s, t)$ egy hálózat, és rajta egy f folyam. Ekkor a $(G' = (V, E'), r, s, t)$ maradékhálózatot a következőképpen kapjuk:*

- *$\forall uv \in E$-re, ha uv telítetlen, akkor uv-t bevesszük az E'-be, előre élnek nevezzük, és az úgynevezett maradék kapacitás $r(uv) := c(uv) - f(uv)$ lesz.*
- *$\forall uv \in E$-re, ha $f(uv) > 0$, akkor vu-t bevesszük az E'-be, vissza élnek nevezzük, és a maradék kapacitás $r(vu) := f(uv)$ lesz.*

3.4. Definíció. *Legyen $S \cup T = V$, ahol $s \in S$ és $t \in T$ teljesül. Ekkor V-nek ezen partícióját (S, T) vágásnak nevezzük. A vágás értéke:*

$$c(S, T) = \sum_{u\in S, v\in T, uv\in E} c(uv).$$

3.1. Tétel. *(Ford-Fulkerson)*
Egy hálózatban a maximális folyam értéke egyenlő a minimális vágás értékével.

3.1. Szintező algoritmus

A Goldbergtől és Tarjantól származó maximális folyamot kereső szintező algoritmust, push-relabel vagy előfolyam algoritmusnak is szokták nevezni [6]. Ellentétben a később említett növelőutas algoritmusokkal, ez a módszer lokális változtatásokból áll, és ahelyett, hogy minden lépésben egy szabályos folyamot tartana fent és addig növelné, amíg maximális nem lesz, itt egy úgynevezett előfolyamot tartunk fent, és akkor áll meg az algoritmus, ha folyamot kapunk.

3.5. Definíció. *Legyen $G = (V, E)$ irányított gráf, ekkor egy $x\colon E \to \mathbb{R}^+$ függvényre és $S \subseteq V$ halmazra, definiáljuk a $\varrho_x(S)$ és a $\delta_x(S)$ értékeket a következőképpen:*
$\varrho_x(S) = \sum(x(uv) : uv \in E, uv \text{ belép } S\text{-be})$.
$\delta_x(S) = \varrho_x(V - S)$.

3.6. Definíció. *A $G = (V, E)$ irányított gráf élhalmazán értelmezett $x\colon E \to \mathbb{R}^+$ függvényt előfolyamnak nevezzük, ha az s pont kivételével minden v csúcsra $\gamma_x(v) \geq 0$, ahol $\gamma_x(v) := \varrho_x(v) - \delta_x(v)$.*

3.7. Definíció. *Egy $v \in V$, $v \neq t$ csúcsot*

- *aktívnak nevezünk, ha $\gamma_x(v) > 0$.*
- *passzívnak nevezünk, ha $\gamma_x(v) = 0$.*

3.8. Definíció. *Egy $e \in E$ élt*

- *csökkenthetőnek hívunk, ha $x(e) > 0$.*
- *növelhetőnek hívunk, ha $x(e) < c(e)$.*

A csúcsokat szintekbe soroljuk egy h szintfüggvény segítségével. Egy v csúcs szintjét $h(v) \geq 0$ jelölje. Az algoritmus során végig $h(t) = 0$ illetve $h(s) = n$, ahol n a csúcsok száma. A további csúcsok szintjei növekedhetnek, de nem csökkenhetnek az algoritmus lépései során. Egy uv élt felmenőnek nevezünk, ha $h(v) > h(u)$, illetve lemenőnek, ha $h(v) < h(u)$. Az algoritmus során minden lépésben adott egy x előfolyam és egy h szintfüggvény, melyekre mindig teljesülnek az úgynevezett illeszkedési feltételek:

- $x(uv) > 0$ esetén $h(v) \leq h(u) + 1$, azaz a csökkenthető élek legfeljebb egy szintet lépnek felfelé.

- $x(uv) < c(uv)$ esetén $h(v) \geq h(u) - 1$, azaz a növelhető élek legfeljebb egy szintet lépnek lefelé.

Más szóval a felmenő csökkenthető uv élekre $h(v) = h(u) + 1$, és a lemenő növelhető uv élekre $h(v) = h(u) - 1$ egyenlőség az algoritmus során végig teljesül. Az ilyen típusú éleket összefoglaló néven feszes éleknek nevezzük.

3.1. Lemma. *Ha az illeszkedési feltételek teljesülnek az x előfolyamra, akkor létezik olyan $0 < i < n$ index, hogy az i. szint, azaz $\{v : h(v) = i\}$ üres. Az i. szint felett lévő csúcsok közé, azaz a $S := \{v : h(v) > i\}$ halmazba nem lép be csökkenthető él és S-ből nem lép ki növelhető él.*

3.1. Következmény. *Ha x folyam, akkor $s \in S \subseteq V - t$ miatt x maximális nagyságú folyam és az S halmaz egy minimális vágást határoz meg.*

3.1.1. Az algoritmus lépései

Kezdetben legyen $h(s) := n$ és minden $v \neq s$ csúcsra $h(v) := n$. Az előfolyamot a felső korlátra állítjuk az összes s-ből induló élen, azaz $x(sv) = c(sv)$, és minden további élre legyen $x(e) = 0$. Az algoritmus minden lépése kétféle lehet, vagy megnöveli egy aktív csúcs szintjét, vagy módosítja egy feszes él előfolyam értékét.

Ha létezik aktív csúcs, akkor válasszuk ki közülük a maximális szintűt és jelöljük v-el:

- Szintemelés:
  Ha nincs se v-be felmenő csökkenthető él, se v-ből lemenő növelhető él, azaz v élei nem feszesek, akkor $h(v)$ értékét növeljük meg eggyel, más szóval v-t rakjuk eggyel magasabb szintre.

- Előfolyam módosítás:
  Ha létezik v-be belépő $e = uv$ feszes él, akkor csökkentjük $x(e)$-t $min\{x(e), \gamma_x(v)\}$ értékkel.
  Ha létezik v-ből kilépő $e = vu$ feszes él, akkor növeljük $x(e)$-t $min\{c(e) - x(e), \gamma_x(v)\}$ értékkel.

Ha nem létezik aktív csúcs, vagyis x előfolyam egyben folyam is, akkor a 3.1 következmény miatt az x maximális folyam és az algoritmus véget ér.

3.2. Tétel. *Az algoritmus legfeljebb $\mathcal{O}(n^3)$ lépés után véget ér.*

3.2. Boykov-Kolmogorov-algoritmus

Yuri Boykov és Vladimir Kolmogorov kidolgoztak egy maximális folyamot kereső algoritmust [7], amely kifejezetten a számítógépes látásban megjelenő esetekre lett kifejlesztre, és az ilyen példákon felülmúlja a szintező algoritmus teljesítményét. A Boykov-Kolmogorov-algoritmus, rövidebben BK-algoritmus a növelőutas folyam algoritmusok közé tartozik. Két nem átfedő fát S-et és T-t tart fent, amelyek gyökerei az s forrásnál, és a t nyelőnél vannak, és élei telítetlen élek. A fákon kívül lévő csúcsokat szabad csúcsoknak nevezzük. A fák csúcsait 2 részre lehet bontani: az aktív csúcsokra, amik a fák leveleiben található csúcsok és a passzív, azaz a kereső fa belsejében lévő csúcsokra. Az aktív csúcsok mentén tudnak növekedni a kereső fák új csúcsokat szerezve a szabad csúcsok közül. Az algoritmus 3 fázist ismételget folyamatosan az alábbi sorrendben:

- növesztési fázis:
  Ebben a fázisban a fák bővülnek. Az aktív csúcsokból induló telítetlen élek mentén új csúcsokat veszünk a fákhoz a szabad csúcsok halmazából. Az újonnan csatlakozott csúcsok aktívvá válnak, és amint egy aktív csúcs összes szomszédját átvizsgáltuk, akkor a csúcs passzívvá válik. A növesztési fázis befejeződik, ha egy aktív csúcsból megy telítetlen él a másik keresőfa egy csúcsába, mert ilyenkor találtunk egy utat a forrástól a nyelőig, ami mentén tudjuk javítani az aktuális folyamot.

- javító fázis:
  Növeli a folyamunkat a növesztési fázis végén talált út mentén a lehető legnagyobb folyamértékkel. Ezáltal az úton legalább 1 él telítődni fog, és így a keresőfáink erdőkre esnek szét azáltal, hogy a fa egyes csúcsai nem lesznek összekötve szüleikkel, mivel bizonyos élek telítetté válnak, az ilyen csúcsokra árvákként hivatkozok a továbbiakban.

- helyreállítási fázis:
  Lényege, hogy helyreállítsa S és T erdőket, hogy újra egy-egy fát alkossanak. Minden árvához megpróbál szülőt találni úgy, hogy a szülőnek és az árvának ugyanabba az S vagy T erdőbe kell tartoznia. Ha telítetlen él mentén nem tudunk szülőt találni neki, akkor átkerül a szabad csúcsok közé és az eddigi gyerekeit árvákká változtatja. A szakasz véget ér, ha nem marad árva csúcs, ekkor az algoritmus folytatódik a növesztési fázissal.

Az algoritmus véget ér, ha a növesztési fázisban az S és T nem tudnak növekedni (nincs több aktív csúcs) és a 2 halmaz között csak telített élek mennek.

3.3. Dinic-algoritmus

Jefim Dinic algoritmusa is a növelőutas folyam algoritmusok családjába tartozik. A Dinic-algoritmus a Ford-Fulkerson-algoritmusban használt maradékhálózat [3.3] helyett szintezett maradékhálózatot használ, amit úgy kapunk, hogy a maradékhálózatban s-ből indított szélességi kereséssel kiszámítjuk a csúcsok s-től való távolságait (jelöljük ezt $d_s(v)$-vel), azaz a szintjeiket, és kitörlünk minden olyan élt a maradékhálózatból, ami nem valamelyik szintről eggyel nagyobb szintre lép, tehát azon uv éleket hagyjuk meg, amire $d_s(u) + 1 = d_s(v)$ teljesül. Ebben a hálózatban olyan f' folyamot keresünk, hogy az f' által telített éleket törölve ne maradjon irányított út az s-ből t-be, az ilyen f' folyamot blokkoló folyamnak nevezzük. A Dinic-algoritmus fázisokban végzi a folyamnöveléseket mindaddig, amíg van irányított út s-ből t-be, és minden fázisban a szintezett maradékhálózat egy blokkoló folyamával növeli az aktuális folyamunkat. A Ford-Fulkerson tétel értelmében ha nem létezik $s \to t$ út a maradékhálózatban, akkor maximális folyamot kaptunk.

3.3. Tétel. *Egy blokkoló folyammal való növelés után a maradékhálózatban szigorúan nő a távolság a forrás és a nyelő között.*

3.4. Tétel. *A Dinic-algoritmus lefut egyszerű megvalósítással $\mathcal{O}(n^2m)$, illetve jobb adatstruktúrával (dinamikus fák segítségével) $\mathcal{O}(nm \log n)$ lépésben.*

Bizonyítás. Csak az $\mathcal{O}(n^2m)$ futásidőt bizonyítom. Mivel $d_s(t)$ minden körben növekszik, a körök száma legfeljebb $n - 1$. Minden kör két szakaszból áll: a szintezett hálózat felépítése és egy blokkoló folyammal való növelés. A szintezett maradékhálózat $\mathcal{O}(m)$ időben szélességi keresés (BFS) segítségével felépíthető. A blokkoló folyamot úgy építjük, hogy mélységi kereséssel $s - t$ utakat keresünk a szintezett maradékhálózatban. A DFS algoritmussal addig haladunk előre, amíg egy olyan csúcshoz jutunk, ahonnan már nem tudunk előre menni (nevezzük ezt blokkoló csúcsnak) vagy, amíg el nem jutunk a t csúcsig, így egy keresés ideje $\mathcal{O}(n)$. Ha egy blokkoló csúcsig jutottunk, akkor ebben a körben már nem megy több $s-t$ út rajta keresztül, tehát elhagyhatjuk a maradékhálózatunkból. Ha a t csúcsig jutottunk, akkor $\mathcal{O}(n)$ időben javítunk az $s - t$ út éleinek folyamértékein. Legfeljebb n-szer törölhetünk blokkoló csúcsot a maradékhálózatból, és a megtalált $s - t$ útvonalak száma legfeljebb m, mert minden talált útvonal legalább egy élt telít, eltávolítva azt a maradékhálózatból. Tehát a mélységi keresések száma legfeljebb $n+m$. Minden körben egy blokkoló folyamot megtalálunk $\mathcal{O}(nm)$ időben és legfeljebb $n - 1$ darab körünk van, ezért a Dinic-algoritmus legfeljebb $\mathcal{O}(n^2m)$ lépést tesz. □

3.4. Módosított Dinic-algoritmus

Az előző fejezetben leírt Dinic-algoritmust a Python nevű programozási nyelvben valósítottam meg, amiben a networkx nevű programcsomag adatszerkezeteit és függvényeit használtam fel. Ezt az algoritmust alakítottam át a Thomas Bläsius, Tobias Friedrich és Christopher Weyand: Efficiently computing maximum flows in scale-free networks című cikkének 3. fejezetében tárgyalt gyorsítási módosítások alapján [1]. A gyorsulások egyrészt annak köszönhetőek, hogy a folyamok és vágások gyakran a hálózat csak egy kis részét érintik.

Az egyik legfontosabb módosítás a szintezett maradékhálózát felépítésében van. Míg a hagyományos algoritmus egy egyszerű szélességi keresést hajt végre, ahelyett használjunk kiegyensúlyozott kétirányú szélességi keresést. A kétirányú BFS algoritmusról belátták, hogy skálafüggetlen hálózatokon futásidejük csúcsszámban nézve szublineáris, azaz nagy valószínűséggel kevesebb a lépésszáma mint a csúcsok száma. Például a hiperbolikus véletlen gráfoknál bebizonyították, hogy nagy valószínűséggel $\mathcal{O}(n^{2-\frac{1}{\alpha}} + n^{\frac{1}{2\alpha}} + \delta_{max})$ futásidejű a kétirányú keresés, ahol az $\alpha \in (\frac{1}{2}, 1)$ paraméter a hatványfüggvényt követő fokszámeloszlás kitevőjétől függ és δ_{max} a maximális fokszámot jelenti.

A kiegyensúlyozott kétirányú BFS-ben az előre menő keresés mindig az s forrásból indul, míg a visszafelé keresés a t nyelőből, és minden egyes lépésben az előre/hátra lépés közül azt preferáljuk, amelynél az aktuális szinten lévő csúcsok száma legkisebb. Egy legrövidebb $s-t$ utat találtunk, ha egy olyan v csúcsot fedeztünk fel, amit már a másik irányból láttunk. A szintezett maradékhálózátunknak minden legrövidebb $s-t$ utat tartalmaznia kell, így még a v csúcs meglátásával nem állíthatjuk le a keresést, hanem az aktuális szint végignézését még be kell fejezni.

Jelölje $d_s(v)$ a v csúcs s-től való távolságát. A következő feladatunk mélységi bejárás (DFS) segítségével $s-t$ utat keresni a szintezett segédgráfban ami hagyományosan azon uv éleken haladna, amire $d_s(u)+1 = d_s(v)$ teljesül, de kétirányú kereséssel nem tudjuk mindegyik v csúcsra a $d_s(v)$ távolságot, mivel a visszafelé keresésnél a $d_t(v)$ távolságok ismertek. Emiatt a DFS algoritmusunkat úgy módosítjuk, hogy azon uv éleket vizsgálja amelyek vagy növelik a forrástól vagy csökkentik a nyelőtől való távolságot, azaz teljesítik a $d_s(u) + 1 = d_s(v)$, és a $d_t(u) - 1 = d_t(v)$ egyenletek valamelyikét.

Ezekkel a módosításokkal a BFS és DFS algoritmusaink várhatóan szublineáris időben megtalálnak egy blokkoló folyamot. A következő körben az új szintezett hálózat létrehozásához mindegyik csúcsra a $d_s(v)$ és a $d_t(v)$ értékeket vissza kell állítani az alapértelmezett értékekre, amit alapvetően úgy oldanánk meg, hogy végigfutunk az összes csúcson és beállítjuk a szükséges értékeket. Ehelyett kihasználva, hogy a szintezett maradékhálózatban a

csúcsok száma tipikusan töredéke a teljes hálózat csúcsszámához képest, megjegyezzük, hogy mely csúcsokat láttuk meg a kétirányú BFS során. Ezáltal kicsit több memóriát használunk a jobb futásidő érdekében.

Azon csúcsokat, amiket a kétirányú keresés utolsó előre lépésénél láttunk meg, de nem fedeztük fel a visszafelé keresésnél, azok a csúcsok biztosan nem részei a legrövidebb utaknak. Jegyezzük meg az utolsó előre keresés szintjét, ezáltal az ilyen típusú csúcsokat ne is nézzük meg a mélységi keresések során, ezzel is csökkentve a vizsgált csúcsok számát.

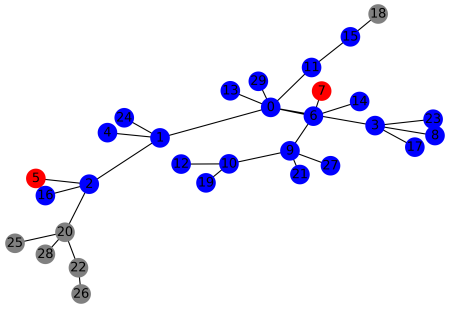
A következőkben megvizsgálom ezen módosítások hatását a futásidőre és a vizsgált csúcsok számára vonatkozóan. Továbbá generált és valós hálózatokon a módosított algoritmus teljesítményét összehasonlítom a korábban említett folyam algoritmusok teljesítményével.

3.5. Algoritmusok tesztelése

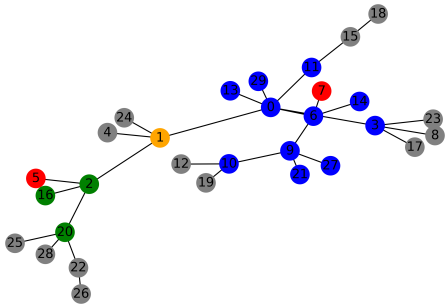
A tesztek során a Python 3.8-as, a NetworkX 2.6.3-as verzióját használtam. A Boykov-Kolmogorov-algoritmus és a szintező algoritmus teljesítményét a networkx.algorithms.flow csomagban található *boykov_kolmogorov* és *preflow_push* függvények segítsével teszteltem. A tesztek programkódjai elérhetőek az alábbi Github felületen:
github.com/PeterKiss18/AlgoritmusokSkalafuggetlenHalozatokban.

3.5.1. BFS és kétirányú BFS összehasonlítása

A legfontosabb módosítás amit a Dinic-algoritmuson végrehajtottam, hogy a szintezett maradékhálózatot kétirányú szélességi bejárással építem fel a hagyományos szélességi bejárás helyett. Ezért először megvizsgálom, hogy ezen bejárások során a meglátott csúcsok száma mennyivel kevesebb kétirányú keresés esetén.



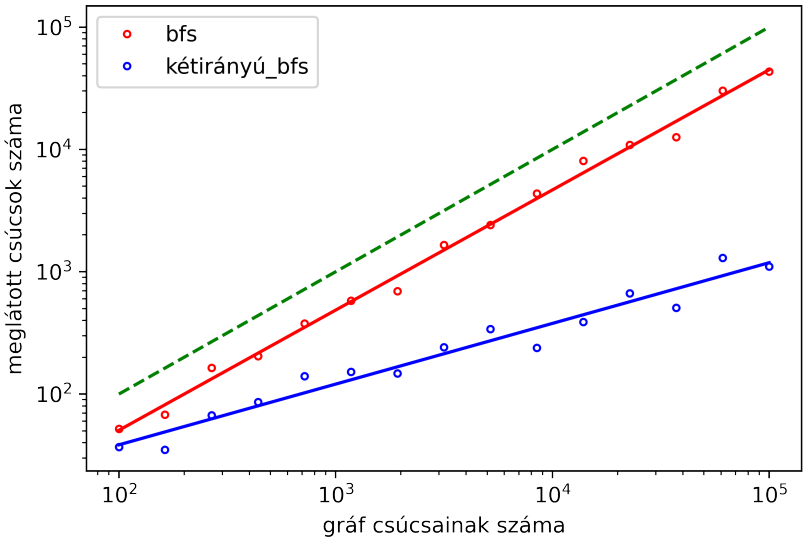
3. ábra. Szélességi keresés egy Barabási-Albert modellből származó gráfon



4. ábra. Kétirányú szélességi keresés egy Barabási-Albert modellből származó gráfon

A 3. ábrán látható egy $n = 30$, $m = 1$ paraméterű Barabási-Albert modellből származó gráfon végrehajtott szélességi keresés eredménye, ahol pirossal jelöltem a kiindulási és a végpontot és kékkel jelöltem a BFS során meglátott csúcsokat. A 4. ábrán ugyanazon a gráfon és ugyanazon kiindulási és végpontokkal lefuttatott kétirányú szélességi bejárás eredménye látható, ahol pirossal jelöltem a kiindulási és a végpontot és kékkel csak az előre menő keresés során meglátott csúcsokat, zölddel a hátrafelé menő keresés során meglátott csúcsokat és narancssárgával a mindkét irányból meglátott csúcsot.

Megvizsgáltam, hogy a Barabási-Albert modellből származó gráfokon a csúcsszám növekedesével a BFS és a kétirányú BFS során hogyan nő a meglátott csúcsok száma. A csúcspárokat (azaz a kezdő- és végpontot) egyenletes eloszlás szerint választottam. Minden gráfon 10 csúcspárra néztem meg a bejárások során meglátott csúcsok számát és ezeket átlagoltam. A kapott eredmény az 5. ábrán látható, ahol a piros pontok a hagyományos szélességi keréses, míg a kék pontok a kétirányú szélességi keresés eredményét prezentálják. A kék illetve piros függvények a log-log skálán feltüntetett pontokra legjobban illeszkedő egyenest mutatják. A zöld szaggatott egyenessel a $h(x) = x$ függvényt ábrázoltam. A tesztben előforduló esetekben a BFS által meglátott csúcsok száma $f(n) = e^{-0.6} \cdot n^{0.982}$ függvénnyel közelíthető (ami a piros egyenesnek felel meg az ábrán), míg a kétirányú BSF által meglátott csúcsok száma $g(n) = e^{1.36} \cdot n^{0.496}$ függvénnyel közelíthető (ami a kék egyenesnek felel meg az ábrán). Ezáltal azt reméljük, hogy maximális folyam keresésnél a szintezett maradékhálózat felépítésének futásidejében és csúcsszámában is jelentős javulást érünk majd el a kétirányú szélességi kereséssel.

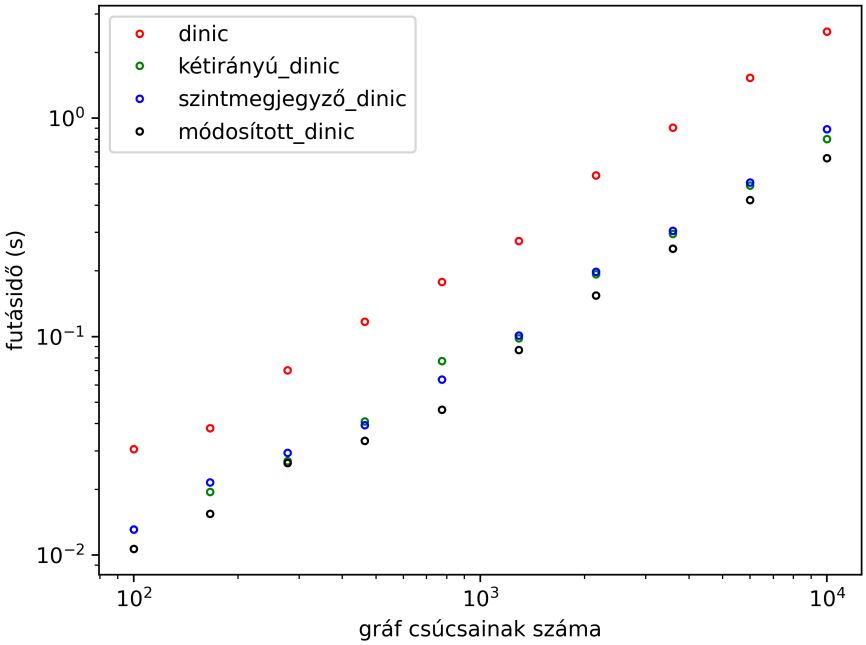


5. ábra. Kétirányú és egyirányú BFS összehasonlítása a BA modellen

3.5.2. Dinic-algoritmus változatainak összehasonlítása

Ebben a részben a módosítások futásidőre gyakorolt hatását vizsgálom. A különböző módosításokat egyesével hajtom végre a Dinic-algoritmuson meghatározott sorrendben, ezáltal 4 algoritmust vizsgálok. Az első változat a hagyományos Dinic-algoritmusnak felel meg. A második változat már kétirányú keresést használt a szintezett maradékhálózat felépítésére. A harmadik változat a másodikkal ellentétben eltárolja, hogy mely csúcsok részei a szintezett maradékhálózatnak. Tehát elég csak ilyen típusú csúcsoknál visszaállítani a $d_s(v)$ és $d_t(v)$ értékeket új szintezett maradékhálózat építése előtt. A negyedik változat a mélységi keresések során nem lép olyan csúcsba, melynek $d_s(v)$ értéke maximális, de nem láttuk meg a visszafelé keresésnél.

A Dinic-algoritmus 4 változatát használom Barabási-Albert modellből származó gráfokon történő maximális folyam keresésre. A csúcspárokat egyenletes eloszlás szerint választottam. Az élekre a kapacitásokat 1-től 30-ig egyenlő valószínűséggel, illetve egymástól függetlenül generáltam. Azt vizsgáltam, hogy az algoritmusok futásidejének aránya hogyan változik a csúcsszám növelésével. Az m paramétert 5-nek választva a csúcsszámot 100-tól 10000-ig növeltem logaritmikus mértékben. Egy n csúcsú és m paraméterű gráfon 10 csúcspárra futtattam mindegyik maximális folyam kereső algoritmust, és a 10 futtatásból átlagos futásidőt számítottam. A kapott eredmények a 6 ábrán láthatók.

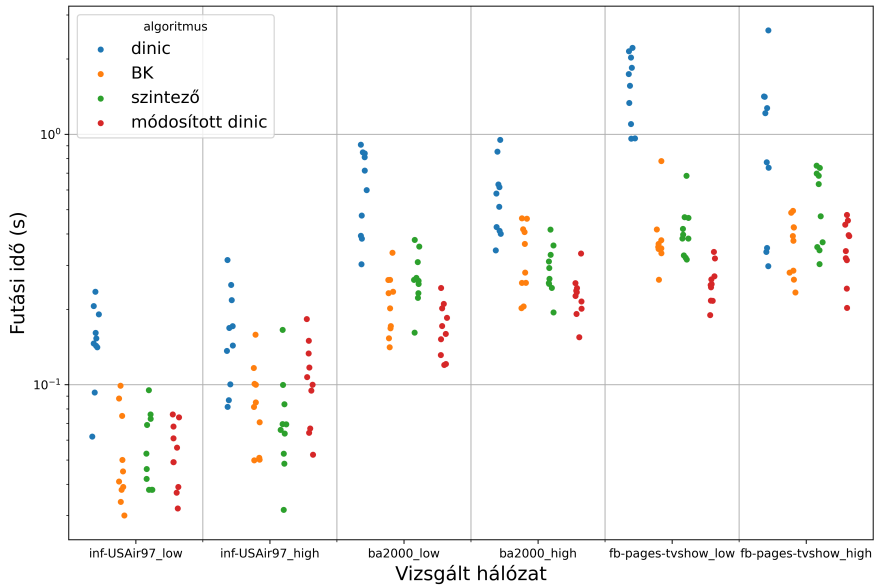


6. ábra. Dinic-algoritmus változatainak összehasonlítása a BA modellen

A teszteredmények azt mutatják, hogy mindegyik módosítás csökkentette az algoritmus futásidejét. A legjelentősebb gyorsulást a kétirányú szélességi keresés bevezetése nyújtotta. Míg a hagyományos Dinic-algoritmusnak átlagosan 2.49 másodpercet vett igénybe egy 10000 csúcsú gráfban egy maximális folyam meghatározása, addig a módosított változatnak átlagosan csak 0.66 másodperc kellett.

3.5.3. Folyam algoritmusok összehasonlítása

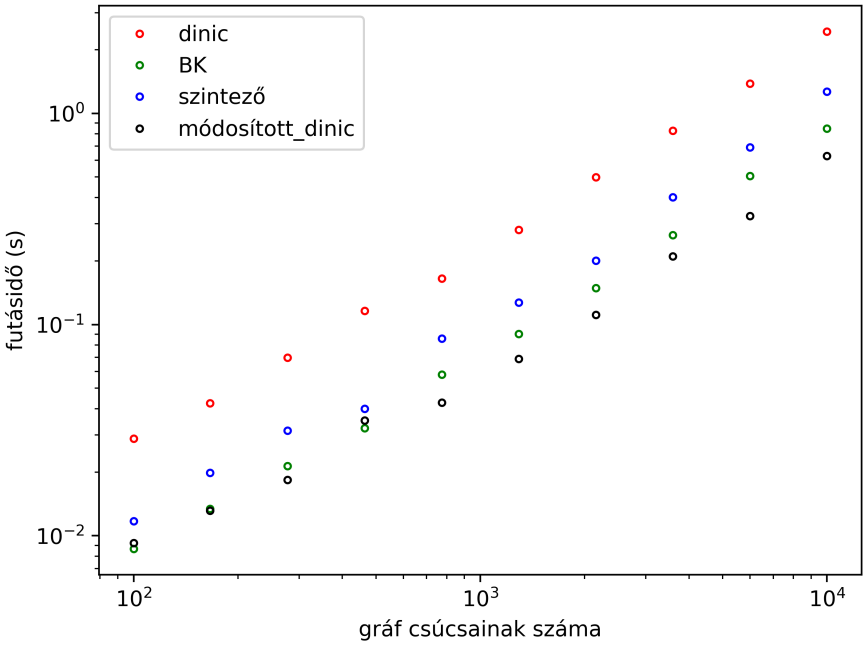
Ebben a részben a Dinic-algoritmus módosított változatát hasonlítottam össze a módosítatlan változat, a Boykov-Kolmogorov-algoritmus, és a szintező algoritmus teljesítményével valódi és generált hálózatokon.



7. ábra. Folyam algoritmusok magas és alacsony fokszámú csúcspárok között

A vizsgálatokat a USAIR97 és a fb-pages-tvshow valós hálózatokon végeztem, amelyek adatait a networkrepository.com-ról szereztem be. Ezen kívűl Barabási-Albert modellből származó gráfokat generáltam a networkx programcsomag *barabasi_albert_graph* függvénye segítségével. Azon hálózatokhoz, melyek élei nem rendelkeztek kapacitásfüggvénnyel, az élekre kapacitásokat 1-től 30-ig egyenlő valószínűséggel generáltam.

A 7. ábrán a USAIR97, a fb-pages-tvshow, és egy $n = 2000$, $m = 7$ paraméterű Barabási-Albert modellből származó gráfokon lefuttatott maximális folyam keresések eredményei láthatóak. Minden pont egy-egy csúcspár közötti maximális folyam keresésnek felel meg. Egy $s - t$ csúcspárt nevezzünk alacsony fokszámú csúcspárnak, ha az s és t csúcs fokszáma az átlagos fokszám legalább 0.5-szerese és legfeljebb 1.5-szerese. Hasonlóan egy $s - t$ csúcspárt nevezzünk magas fokszámú csúcspárnak, ha fokszámaik az átlagos fokszámnak legalább 10-szeresei, de legfeljebb 100-szorosai. Mindegyik gráfhoz 10 db alacsony fokszámú csúcspárt és 10 db magas fokszámú csúcspárt választottam. Az ábráról leolvasható, hogy a 4 algoritmus közül az eredeti Dinic-algoritmus volt a leglassabb mindhárom gráfon alacsony és magas fokszámú csúcspárok esetén is. Ennek ellenére a módosított változat a leggyorsabb volt, és megfigyelhető, hogy alacsony fokszámú csúcspárok esetén nagyobb futási idő javulást ért el, mint a magas fokszámú csúcspárokon.



8. ábra. Folyam algoritmusok futásidő elemzése a BA modellen

A Barabási-Albert modell gráfjain megvizsgáltam, hogy az algoritmusok futásideje hogyan változik a csúcsszám növelésével. A folyam feladatokat a 3.5.2. részben leírtak szerint generáltam. A kapott eredmény a 8. ábrán látható. A kísérletek azt mutatják, hogy a Boykov-Kolmogorov-algoritmus kisebb csúcsszám esetén közel azonos átlagos futási időt produkál, mint a módosított Dinic-algoritmus. A nagyobb csúcsszámú gráfokon ezen kísérletekben is mindig a módosított Dinic-algoritmus érte el a legalacsonyabb futási időt.

Ezen tesztek alapján a módosítások sikeresnek tekinthetőek, hiszen nemcsak az eredeti Dinic-algoritmushoz képest javult a futási idő, hanem a Boykov-Kolmogorov-algoritmus, és a szintező algoritmus teljesítményét is felülmúlta skálafüggetlen hálózatokon.

4. Minimális lefogó csúcshalmaz keresés

A probléma definiálása után, röviden összefoglalom a probléma elméleti nehézségét általános gráfokra nézve. Ezután bemutatok egy $\mathcal{O}(m \log n)$ futásidejű algoritmust, ami hiperbolikus véletlen gráfokon $(1 + o(1))$-közelítést ad $(1 - o(1))$ valószínűséggel.

4.1. Definíció. *Egy $G = (V, E)$ gráf lefogó csúcshalmazának nevezünk egy $B \subseteq V$ csúcshalmazt, ha minden élnek legalább az egyik végpontja B-ben van.*

Egy lefogó csúcshalmaznál bővebb csúcshalmaz is automatikusan lefogó, így a legkevesebb csúcsot tartalmazó halmazt szeretnénk megkeresni, amit minimális lefogó csúcshalmaznak nevezünk, és $\tau(G)$-vel jelöljük az elemszámát.

A minimális lefogó csúcshalmaz megtalálása egy általános gráfban NP-nehéz probléma. Ezáltal nem várható, hogy polinomális futásidőben megtaláljuk a minimális lefogó csúcshalmazt, de adhatunk olyan algoritmust, ami kiválaszt egy lefogó csúcshalmazt, aminek elemszáma nem sokkal nagyobb, mint a minimális elemszám.

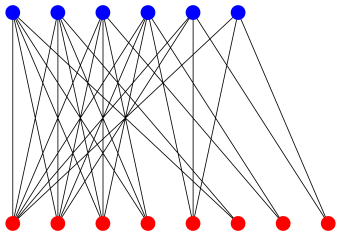
4.2. Definíció. *Legyen adott egy minimalizálási feladat, ahol $X(x)$ jelölje az x input esetén a megengedett megoldások halmazát és legyen adott $f_X : X(x) \to \mathbb{R}$ kiértékelő függvény. A feladatunk azon $y \in X(x)$ megkeresése, amire $f_X(y)$ minimális. Egy D algoritmust α-**közelítőnek** vagy más szóval α-**approximálónak** hívunk ha minden inputra, amelyre létezik megengedett megoldás, visszaad egy y megengedett megoldást, és $f_X(y) \leq \alpha \cdot OPT$, ahol OPT az optimális megoldás értékét jelöli. Az α értéket az algoritmus közelítési hányadosának vagy approximációs hányadosának nevezzük.*

4.1. Standard mohó algoritmus

Az első algoritmus, ami intuitívan eszünkbe juthat, hogy minden lépésben kiválasztunk egy maximális fokú csúcsot, bevesszük a lefogó ponthalmazunkba, letöröljük a csúcsot az éleivel együtt, és ezt a lépést iteráljuk a maradék gráfon mindaddig, amíg van él a gráfunkban. Ezt polinomidőben meg tudjuk valósítani, de nincs olyan c konstans, amire ez az algoritmus c-közelítő lenne.

4.1. Lemma. *A fent leírt algoritmusnak az approximációs hányadosa nem lehet kisebb* $\log(n)$*-nél.*

Bizonyítás. Konstruáljuk meg a $P_n = (L \cup R, E)$ páros gráfot, ahol L csúcshalmaz n darab csúcsból álljon. Legyen $R = \bigcup_{i=2}^{n} R_i$, ahol R_i álljon $\lfloor \frac{n}{i} \rfloor$



9. ábra. Példa egy P_6 gráfra

csúcsból, minden csúcsnak legyen i a fokszáma, és mindegyik különböző L-beli csúccsal legyen összekötve.

A 9. ábrán a kék csúcsok az L csúcshalmaznak, a piros csúcsok az R csúcshalmaznak felelnek meg. Az algoritmus első lépésben az R_n egyetlen csúcsát választja ki, mivel neki a fokszáma n, és az L-beli csúcsoknak legfeljebb $n-1$ a fokszámuk. Tegyük fel, hogy i a legnagyobb olyan index, hogy R_i-beli csúcsunk még van a gráfban. Ekkor az algoritmusunk R_i-beli csúcsot fog kiválasztani, hiszen az L-beli csúcsok fokszámai legfeljebb $i-1$, és minden R_j-beli csúcsnak is kevesebb mint i éle van $j < i$ esetén. Tehát az algoritmusunk a R halmaz összes csúcsát beveszi a lefogó csúcshalmazba, miközben a minimális lefogó csúcshalmaz az L lenne, ami n csúcsot használna.

$$|R| = \sum_{i=2}^{n} |R_i| = \sum_{i=2}^{n} \lfloor \frac{n}{i} \rfloor \geq \sum_{i=2}^{n} (\frac{n}{i} - 1) \geq n \sum_{i=1}^{n} \frac{1}{i} - 2n = n(H_n - 2).$$

Ahol H_n az n-edik harmonikus szám, ami $\log(n)$-el közelíthető, tehát adódik a lemma állítása. □

4.2. Mohó 2-közelítő algoritmus

Az előbbi lemma ellenére a problémára tudunk adni egy egyszerű lineáris futásidejű 2-approximációs algoritmust. Az algoritmusunk minden lépésben válasszon egy tetszőleges élet és mindkét csúcsát vegye be a lefogó csúcshalmazba. Ezután hagyja el a 2 csúcsot a hozzátartozó élekkel, és ismételje mindaddig ezen lépéseket, amíg van él a gráfban.

4.2. Lemma. *Az algoritmus által adott lefogó csúcshalmaz mérete legfeljebb kétszerese a minimális lefogó csúcshalmaz méretének.*

Bizonyítás. Legyen k a kiválasztott élek száma. A kiválasztott k él független, tehát lefogásukhoz legalább k csúcs kell, azaz $\tau(G) \geq k$. Másrészt az algoritmus lefogó csúcshalmazának mérete $2k$. Ezekből következik a lemma állítása, hogy $2k \leq 2\tau(G)$. □

Nem ismerünk $(2-\epsilon)$-közelítő polinomiális algoritmust, ahol $\epsilon > 0$. Van okunk feltételezni, hogy NP-nehéz probléma ilyen közelítést adni, és már bebizonyították, hogy $\sqrt{2}$-közelítést adni NP-nehéz. Ennek ellenére a gyakorlatban a standard mohó algoritmus a valós hálozatokon az optimálishoz nagyon közeli eredményt ad. Felmerül a kérdés, hogy ez a valós hálózatok mely tulajdonságának köszönhető? Mint korábban említettem a valós hálózatok gyakran skálafüggetlennek tekinthetők, így a továbbiakban a skálafüggetlen hálózatok egy speciális modelljével, a hiperbolikus véletlen gráfokkal foglalkozom, és a standard mohó algoritmust módosítva adok egy jobb teljesítményű közelítő algoritmust a minimális lefogó csúcshalmaz problémára.

4.3. Minimális lefogó csúcshalmaz hiperbolikus véletlen gráfban

A 2.1. tétel értelmében elég csak a gráf legnagyobb komponensét vizsgálni, hiszen a többi polilogaritmikus méretű komponensben az összes csúcsrészhalmazt végignézhetjük, és közülük megkereshetjük a minimális lefogót. Ezért a továbbiakban a hiperbolikus véletlen gráf kifejezéssel mindig csak a legnagyobb komponensre hivatkozok.

A 4.4. részben szereplő algoritmus a hiperbolikus véletlen gráfok geometriai tulajdonságait fogja kihasználni. Az algoritmus approximációs hányadosa $(1+o(1))$ lesz asszimptotikusan majdnem biztosan, azaz $(1-o(1))$ valószínűséggel. Az előbbi állítás belátásához, a hiperbolikus véletlen gráfok alábbi tulajdonságainak ismertetése szükséges:

- A hiperbolikus sík területeit kalligrafikus betűkkel szokás jelölni. Legyen $V(\mathcal{A})$ az $\mathcal{A}$ területen lévő csúcsok halmaza. Annak a valószínűsége, hogy egy csúcs az $\mathcal{A}$-ban van:

$$\mu(\mathcal{A}) = \iint_{\mathcal{A}} f(r,\varphi)\, d\varphi\, d\mu, \tag{3}$$

  ahol $f(r,\varphi)$ a 2 polárkoordináta sűrűségfüggvénye a 2. egyenletből.

- Egy u csúcs és v csúcs távolsága nő, ha növeljük a helyvektoraik által bezárt szöget. Jelöje $\theta(r(u),r(v))$ azon maximális szöget, amely esetén még a 2 csúcs össze van kötve (azaz távolságuk legfeljebb R). A $\theta(r(u),r(v))$ felülről becsülhető az alábbi módon:

$$\theta(r(u),r(v)) = 2e^{(R-r(u)-r(v))/2}(1+\Theta(e^{R-r(u)-r(v)})). \tag{4}$$

4.4. Módosított mohó algoritmus HRG-ban

Az algoritmus alapötlete, hogy végigmegy a csúcsokon az r koordinátájuk szerint növekvő sorrendben. Minden lépésben:

- beveszi a soron következő v csúcsot a lefogó csúcshalmazba,
- elhagyja a v csúcsot és a hozzátartozó éleket a gráfból,
- megkeresi a legfeljebb $\tau \log\log(n)$ méretű komponenseket a maradék gráfban,
- ezekben a kicsi komponensekben meghatározza az optimális lefogó csúcshalmazokat.

Később láthatjuk, hogy a τ konstans paraméterrel szabályozhatjuk a kompromisszum mértékét a futásidő és a közelítés minősége között. A legfeljebb $\tau \log\log(n)$ méretű komponensekre a továbbiakban kicsi kompenensekként hivatkozok.

4.5. Algoritmus közelítési hányadosa

Azon csúcsok halmazát amelyeket mohó módon vettünk be a lefogó csúcshalmazunkba jelöljük V_{moho}-val, míg a kicsi komponensekből kiválasztott csúcsok halmazát jelöljük C_{pontos}-al. Triviálisan látszik, hogy a kiválasztott csúcsok halmaza, azaz $V_{moho} \cup C_{pontos}$ lefogó csúcshalmazt alkot. Jelölje $G[V_{pontos}]$ azon részgráfot, amelyet G-ből kapunk V_{moho} csúcsok elhagyásával. Legyen C_{opt} a G gráfunk egy minimális lefogó csúcshalmaza. Ekkor az algoritmus approximációs hányadosa definíció szerint:

$$\delta = \frac{|V_{moho}| + |C_{pontos}|}{|C_{opt}|}.$$

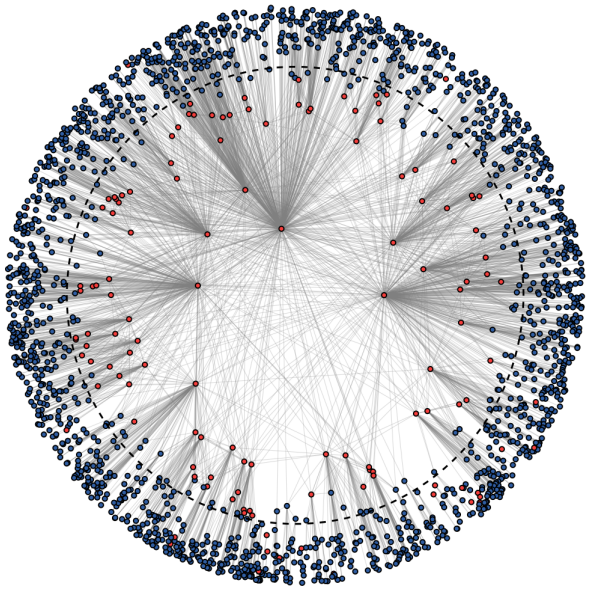
Egy lefogó csúcshalmaz a gráf tetszőleges részgráfján is lefogó csúcshalmaz, így a C_{opt} azon csúcsai, amelyek benne vannak $G[V_{pontos}]$-ban lefogó csúcshalmazt alkotnak. A C_{pontos} csúcshalmaz a $G[V_{pontos}]$ gráf egy minimális lefogó csúcshalmaza, tehát $|C_{pontos}| \leq |C_{opt}|$. Következésképpen

$$\delta \leq 1 + \frac{|V_{moho}|}{|C_{opt}|}.$$

Tehát a cél megmutatni, hogy $|V_{moho}| \in o(|C_{opt}|)$, amiből következne az $(1 + o(1))$ approximációs hányados.

A $|V_{moho}|$ megfelelő becsléséhez válasszunk ki egy időpontot, ami után csak kevés csúcsot veszünk be mohó módon, és a csúcsok többsége már kisebb

komponensekben helyezkedik el. Mivel az algoritmus a csúcsokat r koordinátájuk szerint növekvő sorrendben dolgozza fel, ezért ezen időpont egy ρ küszöbszámmal azonosítható, hogy az adott időpontig azon pontok kerültek feldogozásra melyek r koordinátájára $r < \rho$ teljesül. Ezen ρ értékre a továbbiakban küszöbsugárként hivatkozok. A hiperbolikus körlapot egy belső körre és egy külső sávra bontsuk fel aszerint, hogy az origótól való távolság ρ-nál kisebb vagy nagyobb. Egy ilyen felbontást ábrázol a 10. ábra[1], ahol a V_{moho} csúcsai pirossal míg a többi csúcs kékkel van jelölve. A $|V_{moho}|$ értéket felülről lehet becsülni a belső körben lévő csúcsok számának, és azon csúcsok számának összegeként, amelyek bár a külső sávban vannak, mégis a belső körben lévő csúcsok elhagyása után is $\tau \log \log n$-nél nagyobb méretű komponensben helyezkednek el.



10. ábra. A hiperbolikus véletlen gráf felbontása egy ρ küszöbsugárral

A külső sávban szeretnénk felülről becsülni azon csúcsok számát, amelyek $\tau \log \log n$-nél nagyobb méretű komponensben találhatók, ezért felosztjuk a külső sávot kisebb részekre. Célunk felbontani a hiperbolikus kört azonos γ középponti szögű körcikkekre (ezáltal a külső sávot úgynevezett szektorokra), hogy a külső sávban egyetlen egy él se nyúljon túl egy üres szektoron a 11. ábrán[1] látható módon. A $|V_{moho}|$ becsléséhez a ρ küszöbsugár és γ szög

[1]A 10. és a 11. ábrát Thomas Bläsius, Tobias Friedrich, és Maximilian Katzmann [8] cikkéből kölcsönöztem.

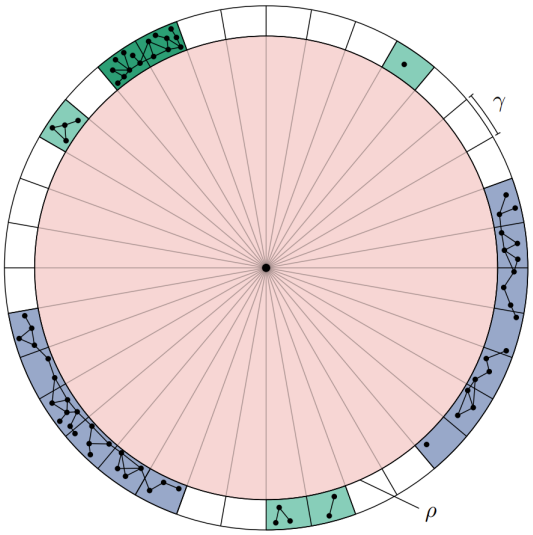
megfelelő megválasztása szükséges. A $\gamma(n, \tau)$ függvényt definiáljuk a következőképpen:

$$\gamma(n, \tau) = \log\left(\frac{\tau \log^{(2)}(n)}{2(\log^{(3)}(n))^2}\right),$$

ahol $\log^{(i)}(n)$ jelentse i darab logaritmus függvény egymásba ágyazását (például $\log^{(2)}(n) = \log\log(n)$).
Ekkor a ρ küszöbsugarat válasszuk meg az alábbi módon:

$$\rho = R - \log(\frac{\pi}{2} \cdot e^{-\log(k)} \cdot \gamma(n, \tau)), \tag{5}$$

ahol $R = 2\log\frac{n}{k}$ a hiperbolikus körlap sugara. A τ szöget a lehető legkisebbre szeretnénk választani amit ρ megenged, így legyen $\tau = \theta(\rho, \rho)$, ami azon maximális szög, amely esetén még két csúcs össze van kötve ha r koordinátájuk legfeljebb ρ. Ekkor a 4. egyenlet értelmében $\gamma = \pi \cdot \gamma(n, \tau)/n \cdot (1 + o(1))$ teljesül.



11. ábra. A hiperbolikus véletlen gráf felosztása körcikkekre

4.5.1. Belső körben lévő csúcsok száma

A belső kört jelöljük $\mathcal{I}$-vel. A belső körben a mohón választott csúcsok száma felülről becsülhető az összes $\mathcal{I}$-ben lévő csúcs számával. Utóbbira a következő lemma ad felső becslést.

4.3. Lemma. *Legyen G egy n csúcsú véletlen hiperbolikus gráf, ahol $\beta = 2\alpha + 1$ a fokszámeloszlás hatványfüggvényének a kitevője. Ekkor nagy valószínűséggel az $\mathcal{I}$ területen lévő csúcsok száma:*

$$\mathcal{O}(n \cdot \gamma(n,\tau)^{-\alpha}).$$

A lemma bizonyításához az $\mathcal{I}$ területen lévő csúcsok számának várható értékét kell felülről becsülni a területének csúcsszámszorosával. Ezután a Chernoff-egyenlőtlenség következményéből adódik a lemma állítása.

Mivel $\gamma(n,\tau) = \omega(1)$, ezért a lemma következménye, hogy nagy valószínűséggel szublineáris nagyságrendű csúcsot választunk ki mohó módon a belső körben.

4.5.2. Külső sáv vizsgálata

A külső sávot mint már korábban említettük γ középponti szögű szektorokra osztjuk fel. Üres szektoroknak nevezzük azon szektorokat, melyekben nincs csúcsa a gráfnak. A γ értékét úgy választottuk, hogy a külső sáv két csúcsa közötti él ne nyúljon túl üres szektoron. Nevezzük láncoknak a nem üres szektorok maximális sorozatát, amelyek kék és zöld színnel jelennek meg a 11. ábrán. Egy láncban több komponens is lehet, így egy lánc csúcsszámát megkaphatjuk a benne található komponensek méreteinek összegeként.

Annak bizonyítására, hogy csak néhány csúcs van $\tau \log\log(n)$-nél nagyobb komponensekben, becsüljük meg azon láncok csúcsszámát, amelyek több, mint $\tau \log\log(n)$ csúcsot tartalmaznak. Ezek a láncok két osztályba sorolhatók. Egy lánc sok csúcsot tartalmazhat, ha középponti szöge túl nagy, azaz sok szektorból áll. Ennek a valószínűsége kicsi, mivel a γ annyira kicsire van választva, hogy a szektorok is elég kicsik, ezáltal nagy valószínűséggel üresek. A másik lehetőség, hogy bár a lánc középponti szöge kicsi, de méretéhez képest túl sok csúcsot tartalmaz. A gráf csúcsait egyenletes eloszlás szerint választjuk a körlapon, így kicsi annak valószínűsége, hogy túl sok csúcsot választanánk a kicsi láncok területén. Ezen állítások precízzé tételéhez, vezessünk be egy w küszöbszámot. Széles láncoknak nevezzük azon láncokat, amelyek több, mint w szektort tartalmaznak és keskeny láncoknak melyek kevesebb, mint w szektorból állnak. A w küszöbértéket úgy szeretnénk beállítani, hogy kicsi legyen annak a valószínűsége, hogy egy lánc széles illetve, hogy egy keskeny lánc több, mint $\tau \log\log(n)$ csúcsot tartalmaz. Legyen $w = e^{\gamma(n,\tau)\cdot\log^{(3)}(n)}$.

Jelöljük $\mathcal{W}$-vel a széles láncok unióját. Legyen $\mathcal{N}$ a hiperbolikus körlap azon keskeny láncok alkotta része, melyek legalább $\tau \log\log(n)$ csúcsot tartalmaznak. Jelöljük N-el az $\mathcal{N}$-ben lévő csúcsok számát. A következő lemmák

$|V(\mathcal{W})|$ és N értékekre adnak felső becslést, amiket nem bizonyítok, de részletes bizonyításaik a már korábban említett [8] cikkben szerepelnek.

4.4. Lemma. *Legyen G egy n csúcsú hiperbolikus gráf. Ekkor nagy valószínűséggel a széles láncokban lévő csúcsok száma:*

$$|V(\mathcal{W})| = \mathcal{O}\left(\frac{\tau^{\frac{3}{4}} \cdot n}{(\log^{(2)}(n))^{\frac{1}{4}} \cdot (\log^{(3)}(n))^{\frac{1}{2}}}\right).$$

4.5. Lemma. *Legyen G egy n csúcsú hiperbolikus gráf. Ekkor nagy valószínűséggel a $\mathcal{N}$-beli keskeny széles láncokban lévő csúcsok száma:*

$$N = \mathcal{O}\left(\frac{\tau \cdot n \cdot \log^2(n)}{\gamma(n,\tau)(\log(n))^{\frac{\tau}{18}}}\right).$$

4.6. Algoritmus helyességére vontakozó tételek

4.1. Tétel. *Legyen G egy n csúcsú véletlen hiperbolikus gráf, ahol $\beta = 2\alpha+1$ a fokszámeloszlás hatványfüggvényének a kitevője. Továbbá legyen adott egy τ konstans. Ekkor a fent definiált algoritmus $\mathcal{O}(m\log(n)^\tau + n\log(n))$ időben megad egy lefogó csúcshalmazt, és az algoritmus közelítési hányadosa $(1 + \mathcal{O}(\gamma(n,\tau)^{-\alpha}))$ aszimptotikusan majdnem biztosan.*

Bizonyítás.
I. Futásidő bizonyítása:
Először is az algoritmus sorba rendezi a gráf csúcsait a sugaraik alapján növekvő sorrendbe, ami $\mathcal{O}(n\log(n))$ időt vesz igénybe. Ezután végigmegyünk a csúcsokon ilyen sorrendben és minden v csúcsra végrehajtjuk a következő lépéseket:

1. Hozzáadjuk v-t a lefogó csúcshalmazunkhoz.
2. Eltávolítjuk a csúcsot a hozzátartozó éleivel a gráfból.
3. Megkeressük azon komponenseket, amik legfeljebb $\tau\log\log n$ méretűek.
4. Ezen komponensekben meghatározzuk pontosan a minimális lefogó csúcshalmazt.

Az első két lépés $\mathcal{O}(1)$ és $\mathcal{O}(\deg(v))$ futásidejű, de a kicsi komponensek megtalálása több időt vesz igénybe. Egy v csúcs eltávolításával legfeljebb $\deg(v)$ darab új komponens jön létre, amiket át kell vizsgálni, hogy legfeljebb $\tau\log\log n$ méretűk-e. Ezen komponenseken szélességi bejárás hajt végre,

ami egyből megáll, ha már több, mint $\tau \log\log n$ csúcsot látott meg. A kicsi komponensek legfeljebb $(\tau \log\log n)^2$ élt tartalmaznak, tehát egy BFS $\mathcal{O}((\log\log n)^2)$ időt vesz igénybe. Egy k csúcsú gráfban egy minimális lefogó csúcshalmazt $1.1996^k \cdot k^{\mathcal{O}(1)}$ lépésben pontosan meg lehet határozni [9]. Mivel egy komponens mérete legfeljebb $n_c = \tau \log\log n$, így $1.1996^{n_c} \cdot n_c^{\mathcal{O}(1)}$ időben megtalálható benne egy minimális lefogó csúcshalmaz. Ez felülről becsülhető $\mathcal{O}(e^{n_c}) = \mathcal{O}(\log(n)^\tau)$ futásidővel, hiszen $n_c^{\mathcal{O}(1)} = \mathcal{O}((e/1.1996)^{n_c})$. Tehát a 4.lépés legfeljebb $\deg(v)$ komponenssel dolgozik és minden komponensben egy minimális lefogó csúcshalmaz megkeresése legfeljebb $\mathcal{O}(\log(n)^\tau)$ időt igényel.

Jelölje $T(n, m, \tau)$ az algoritmus teljes futásidejét. Ezen értéket kapjuk, ha összeadjuk a fent leírt 4 lépés futásidejét minden csúcsra:

$$\begin{aligned} T(n,m,\tau) = \sum_{v\in V} \mathcal{O}(1) + \mathcal{O}(\deg(v)) + \deg(v)\cdot\mathcal{O}((\log\log(n))^2) + \\ +\deg(v)\cdot\mathcal{O}((\log(n))^\tau) = \mathcal{O}((\log(n))^\tau \cdot \sum_{v\in V}\deg(v)) = \mathcal{O}(m(\log(n))^\tau) \end{aligned} \tag{6}$$

II. Közelítési hányados bizonyítása: Korábban láttuk, hogy az algoritmus közelítési hányadosát az alábbi hányados adja meg:

$$\delta = \frac{|V_{moho}| + |C_{pontos}|}{|C_{opt}|}$$

Ezt a 4.5. alfejezet elején felülről becsültük a $\delta \leq 1 + |V_{moho}|/|C_{opt}|$ egyenlőséggel. A $|V_{moho}|$ megfelőle becsléséhez egy ρ küszöbsugár és egy γ szög segítségével a körlapot felosztottuk az $\mathcal{I}$ belső körre, a $\mathcal{W}$ széles láncok területére, és az $\mathcal{N}$ nagy csúcsszámú keskeny láncok területére. Ezen felosztás alapján az approximációs hányadost a következőképpen becsülhetjük:

$$\delta = 1 + \frac{|V(\mathcal{I})| + |V(\mathcal{W})| + |V(\mathcal{N})|}{|C_{opt}|} \tag{7}$$

A $|V(\mathcal{I})|$, $|V(\mathcal{W})|$ és $|V(\mathcal{N})|$ értékeket felülről tudjuk becsülni, a 4.3., 4.4. és 4.5. lemmák alapján. Illetve egy hiperbolikus gráfban a minimális lefogó csúcshalmaz méretére a $|C_{opt}| = \Omega(n)$ becslés aszimptotikusan majdnem biztosan teljesül [10]. Ezeket felhasználva a 7. egyenlőségből kapjuk, hogy:

$$\delta = 1 + \mathcal{O}\left(\frac{1}{\gamma(n,\tau)^\alpha} + \frac{\tau^{\frac{3}{4}}}{\log^{(2)}(n)^{\frac{1}{4}}\cdot\log^{(3)}(n)^{\frac{1}{2}}} + \frac{\tau\cdot\log^2(n)}{\gamma(n,\tau)\log(n)^{\frac{\tau}{18}}}\right). \tag{8}$$

Mivel $\gamma(n,\tau) = \mathcal{O}(\log^{(3)}(n))$, ezért az első összeadandó tag dominál aszimptotikusan. Tehát az algoritmus közelítési hányadosa $(1+\mathcal{O}(\gamma(n,\tau)^{-\alpha}))$ aszimptotikusan majdnem biztosan. □

4.2. Tétel. *Legyen G egy n csúcsú véletlen hiperbolikus gráf, ahol $\beta = 2\alpha+1$ a fokszámeloszlás hatványfüggvényének a kitevője. Továbbá legyen adott egy τ konstans. Ekkor $\mathcal{O}(m\log(n))$ időben megadható egy lefogó csúcshalmaz, aminek közelítési hányadosa $(1+o(1))$ aszimptotikusan majdnem biztosan.*

Bizonyítás. A 4.1. tétel értelmében olyan lefogó csúcshalmazt tudunk keresni $\mathcal{O}(m\log(n)^\tau+n\log(n))$ időben, melynek mérete legfeljebb $(1+\mathcal{O}(\gamma(n,\tau)^{-\alpha}))$-szerese az optimumnak. A $\tau := 1$ választással adódik a tétel futásidőre vonatkozó része. Továbbá $\gamma(n,1) = \omega(1)$ és $\alpha \in (\frac{1}{2}, 1)$, ezért a közelítési hányadosra $(1+o(1))$ adódik. □

4.7. Algoritmus használata valós hálózatokon

Az előbb bemutatott algoritmus a hiperbolikus véletlen gráfokon a csúcsok feldolgozási sorrendjét a sugaraik alapján határozza meg, amelyek nem ismertek a valós hálózatoknál. A hiperbolikus véletlen gráfokban azonban erős korreláció van egy csúcs sugara és foka között. Ezért az algoritmus lépéseit utánozhatjuk valós hálózatoknál úgy, hogy a csúcsokat fokszámuk alapján csökkennő sorrendben dolgozzuk fel. Ezáltal a standard mohó algoritmus egy módosított verzióját kapjuk azzal a változtatással, hogy a kicsi komponensekben pontosan megkeressünk egy minimális lefogó csúcshalmazt.

Thomas Bläsius, Tobias Friedrich, és Maximilian Katzmann az Efficiently approximating vertex cover on scale-free networks with underlying hyperbolic geometry című cikkükben 42 valós hálózaton mérték össze a standard mohó és a 4.4. alfejezetben bemutatott algoritmus módosított változatának teljesítményét [8]. A kicsi komponensek küszöbértéknek a $10\log\log(n)$ értéket választották, ami a $\tau = 10$ esetnek felel meg. A kísérleteik azt mutatták, hogy az átlagos közelítési hányados $1,009$-ről $1,004$-re csökkent, ha a standard mohó algoritmus helyett, a kicsi komponensekben pontosan határozták meg a minimális lefogó csúcshalmazok méretét. Jelölje $|V_{moho}|$ a standard mohó algoritmus által adott lefogó csúcshalmaz méretét, $|V_{jav}|$ a módosított algoritmus által adott lefogó csúcshalmaz méretét és $|V_{opt}|$ egy minimális lefogó csúcshalmaz méretét. Ekkor a relatív hiba jelentse a $\dfrac{|V_{jav}| - |V_{opt}|}{|V_{moho}| - |V_{opt}|}$ hányadost. Ezáltal csak azokon a hálózatokon érdemes vizsgálni a relatív hibát, amelyeken a standard mohó algoritmus nem egy minimális lefogó csúcshalmazt talált meg. Ezeken a hálózatok a relatív hibák átlagára $0,39$ értékét kaptak eredményül. Azaz a bemutatott algoritmus átlagosan $0,39$-szeresére csökkentette azon csúcsok számát, amelyekkel a standard mohó algoritmus meghaladta az optimális megoldást.

5. Összegzés

Konklúzióként levonható, hogy az általános gráfokon legjobban teljesítő algoritmusok a való életben fellelhető hálózatokon nem feltétlen jobbak a lassabbnak hitt algoritmusoknál. Láttunk példát olyan közelítő algoritmusra, amelytől általános gráfokon nem várható elfogadható eredmény, de a valódi hálózatokon mégis az optimumhoz nagyon közeli eredményt kapunk.

Megfigyeltük, hogy skálafüggetlen hálózatok modelljeiben 2 csúcs közötti legrövidebb utat sokkal hatékonyabb kétirányú szélességi kereséssel meghatározni, mint a hagyományos szélességi kereséssel, mint ahogy úthálózatokra is a kétirányú Dijkstrát használják. Ezen ötlet alapján módosításokat végeztünk a Dinic-algoritmuson. Ezáltal egy olyan algoritmust kaptunk, ami gyorsabban talált maximális folyamot skálafüggetlen hálózatokban, mint az általános gráfokon leggyakrabban használt folyam kereső algoritmusok.

Végezetül a hiperbolikus véletlen gráfokon láttunk egy közelítő algoritmust a minimális lefogó csúcshalmaz problémára, ami $O(m \log(n))$ időben $(1 - o(1))$ valószínűséggel megadott egy $(1 + o(1))$-közelítést. Az algoritmus a standard mohó megközelítést használta a hiperbolikus véletlen gráfok geometriai tulajdonságával. A 4.7. alfejezetben elhangzottak alapján kijelenthetjük, hogy ezen algoritmus apróbb módosításokkal a való életbeli hálózatokon is hatékonyan használható.

Ábrák jegyzéke

Hivatkozások

[1] Thomas Bläsius, Tobias Friedrich, and Christopher Weyand: Efficiently computing maximum flows in scale-free networks. CoRR, abs/2009.09678, 2020.

[2] Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá: Hyperbolic geometry of complex networks. Physical Review E, 82:036106, 2010.

[3] Tobias Friedrich and Anton Krohmer: On the Diameter of Hyperbolic Random Graphs. SIAM Journal on Discrete Mathematics, 32(2):1314–1334, 2018.

[4] Barabási, Albert László and Albert, Réka: Emergence of scaling in random networks. Science, 286:509–512, 1999.

[5] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. Network flows: Theory, algorithms and applications. Prentice-Hall, Inc., 1993.

[6] Frank András: Kombinatorikus algoritmusok II. 2.2 alfejezet, 2009.

[7] Y. Boykov and V. Kolmogorov: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. IEEE Transactions on Pattern Analysis and Machine Intelligence, 26(9):1124—1137, 2004.

[8] Thomas Bläsius, Tobias Friedrich, and Maximilian Katzmann: Efficiently approximating vertex cover on scale-free networks with underlying hyperbolic geometry. CoRR, abs/2010.02787, 2020.

[9] Mingyu Xiao and Hiroshi Nagamochi: Exact algorithms for maximum independent set. Information and Computation, 255:126–146, 2017.

[10] Ankit Chauhan, Tobias Friedrich, and Ralf Rothenberger: Greed is Good for Deterministic Scale-Free Networks. In 36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, pages 33:1–33:15, 2016.