A szűrő elhelyezés probléma és alkalmazása a tartalom duplikációk megszünetetésére

Varga Péter

2012. november 27.

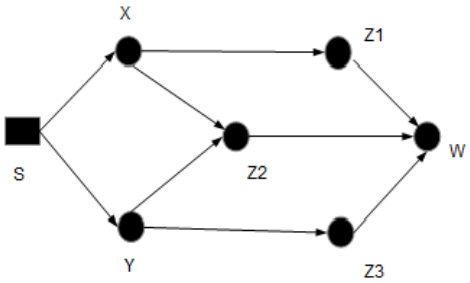
Tanulmány: Dora Erdös,Vatche Ishakian, Andrei Lapets, Evimaria Terzi, Azer Bestavros Filter-Placement Problem and its Application to Minimizing Information Multiplicity cikke alapján
ELTE-IK Msc. 2012 őszi félév

1. Absztrakt

Számos információs hálózatban, az adatok (például frissítések a közösségi hálózatokban, hírek terjedése összekapcsolt RSS-hírfolyamokban, útvonalfrissítések ad-hoc hálózatokban, stb.) összehangolatlan módon terjednek: a csomópontok gyakran minden információt továbbítanak a szomszédaiknak, függetlenül attól, hogy ezt az információt a szomszéd megkapta-e már mástól. Ez a fajta adatterjesztés eredményezhet jelentős, mégis szükségtelen kommunikációt, feldolgozást, így csökkentve az információs hálózatok hatékonyságát. A káros hatások enyhítésére, azt javasolják a cikk írói, hogy a csomópontok egy részhalmaza végezzen szűrést, azaz távolítsák el, vagy jelentősen csökkentsék a duplikált adatokat, amik rajtuk keresztül haladnak. Ezekre a csomópontokra filterekként fognak hivatkozni. Formálisan definiálják a Filter Placement problémát, ami egy kombinatorikai optimalizációs probléma, és tanulmányozzák a probléma számítási komplexitását különböző tipusú gráfokon. Továbbá bemutatnak polinom idejű algoritmusokat a probléma megoldására. A kísérleti eredmények arra utalnak, hogy viszonylag kis számú filter beépítésével is meglehetősen hatékonyan távolíthatóak el a duplikált adatok.

2. Bevezető

Információs hálózatokkal találkozhatunk számos alkalmazás, közösségi hálózat, szenzorhálózat, ad-hoc hálózat és más hálózat során. Egy információs hálózatban a tartalom a tartalom gyártójától, például a forrástól, terjed a fogyasztókig irányított linkeken, amik összekötik a csomópontokat a hálózatban. Egy információs hálózat hasznosságát jól lehet mérni a rajta keresztülfolyó információ terjedésének hatékonyságával. Egy hálózat akkot tekinthető jól működőnek, ha az összes csomópont up-to-date, és az újonnan generált adatok terjednek rajta.
`Motiváció:` A legtöbb információs hálózat közös tulajdonsága az, hogy a tartalom terjedése nem koordinált: a csomópontok minden információt, amit megkapnak továbbítanak a szomszédaiknak, attól függetlenül, hogy ezt az információt azok megkapták-e már. Ennek a stratégiának a következménye lehet a csomópont autonómia, korlátozott képességek és hiányzó helyi erőforrások, vagy a hiányzó routing információk. A koordinálatlan duplikálás eredményeként a csomópontok egy információs hálózatban többször is megkaphatják ugyanazt az információt a hálózat különböző útjain. Az ezáltal okozott redundancia jelentős, szükségtelen kommunikácót és feldolgozási overheadet jelent. Az alábbi ábrán keresztül látható az előbb leírt probléma. Ebben a pédában az S a forrás, és az információnak az X és az Y csomópontokhoz kell eljutnia. Az összes többi csomópont csupán továbbítja az információt az irányított éleken keresztül, más szerepük nincs.



1. ábra: Működés a legegyszerűbb (simple) esetben

Könnyedén észrevehető, hogy egy i üzenet eléri X-t és Y-t S-ből. Az X és Y csomópontok továbbítják i-t, így i-t megkapja $Z1$, $Z2$ és $Z3$ is. Tehát $Z2$ i-t kétszer kapja meg, egyszer X-től, egyszer Y-tól. Ezután $Z1, Z2, Z3$ továbbítanak mindent amit megkaptak W-nek, ezáltal W megkap $(1 + 2 + 1)$ másolatot i-ből. Az viszont egyértelmű, hogy W-nek elég lett volna egy másolat i-ből. A negatív hatások enyhítésére a csomópontok egy halmazát a cikk írói kiválasztják és felszerelik egy „de-duplication" nevű funkcióval, ami a duplikációkat megszünteti. Ezeket a csúcsokat filtereknek hívják, amik sosem küldik el ugyanazt az adatot kétszer ugyanannak a szomszédnak. Filter Placement problémának nevezik a csomópontok egy olyan halmazának a kiválasztását, amelyek filterként fognak működni. Továbbá fontos megjegyezni, hogy a filterek elhelyezése más módon nem hat a hálózatra, a filterek csupán a rajtuk keresztülmenő duplikált adatokat szűrik ki. A példában két filter elhelyezése ($Z2$ és W pozíciókban) elegendő a redundancia teljes megszüntetésére. Ugyanakkor túl sok filter beépítése szükségtelen feldolgozást eredményezhet [3][4][5].

3. A Filter Placement probléma

Legyen egy hálózat összekapcsolt entitások halmaza, továbbá az entitások továbbítják az információt egymásnak. Egy ilyen hálózatot egy irányított $G(V, E)$ gráffal reprezentálnak, amit kommunikációs gráfnak (c-gráf) is hívnak. A hálózatban résztvevő entitások tartoznak a csúcsok halmazába. Egy irányított (u,v) $\in E$ él a gráfban egy linket reprezentál, aminek a segítségével u adatokat tud terjeszteni v-nek. G néhány csúcsa új adatot tud létrehozni, ezeket a csúcsokat source-oknak, azaz forrásoknak neveznek. Amint egy új adat létrejön a következőképpen terjed G-ben: minden csomópont, ami megkap egy példányt az adatból továbbítja azt a kimenő élein. Mivel a terjesztés vak, ezért a csomópont egy adatból több példányt is megkaphat, ami redundanciához vezet.
Az információ terjesztési modell determenisztikus abban az értelemben, hogy

minden adat, ami elér egy csomópontot továbbítva van az adott csúcs minden szomszédjának. A modellben bemutatott G-t lehetne használni több különböző i adatra is, itt most csupán egy elemre koncentrálnak, az eredmények azonosak lennének különbező i adatokra is.

3.1. Filterek

Legyen i egy új adat az S forrásból. Ahhoz, hogy i eljusson a v csúcshoz, egy irányított úton kell végighaladnia S-ből V-be. Mivel több út is van S-ből V-be, V több másolatát is meg fogja kapni a csomagnak. Továbbá, ha V-nek van gyermeke, akkor minden gyermekének is továbbítani fogja i minden egyes másolatát is. Ahhoz hogy csökkenjen a redundancia, egyes csúcsokat kiegészítenek szűrő funkcióval. Filterként fognak azokra a csúcsokra hivatkozni, amelyek ezzel a funkcióval rendelkeznek.A filter egy függvény, ami bementre megkap egy adatokból álló I multihalmazt, és a kimenete egy I' halmaz, I' számossága kisebb mint I számossága. A szűrő funkció kiválasztása nagyban függ, hogy hol, milyen hálózatban használjuk azt. Az egyszerűség kedvéért rögzítik ezt a funkciót, úgy hogy a függvény minden duplikátumot kiszűr: a szűrő csúcs végrehajt egy ellenőrzést minden bejövő adatra, hogy továbbította-e már azt korábban. Ha nem, akkor minden szomszédjának továbbküldi azt. Egy filter csomópont sosem továbbít már továbbított adatot. Legyen $v \in V$ egy tetszőleges csúcs a gráfban. Definiálják $\phi(0, v)$ -t, azon adatok számát amit a v csúcs fogad, amikor nincsenek szűrők elhelyezve. Legyen $A \subseteq V$, ekkor $\phi(A, v)$ jelölje a v csomópont által fogadott adatok számát, amikor a szűrők az A halmaz csúcsai. Legyen $X \subseteq V$ és legyen $\phi(A, X) = \sum_{x \in X} \phi(A, x)$. Egy adott i elem esetén az összes üzenetek száma, amit V-ben lévő csúcsok fogadnak szűrők hiányában a $\phi(0, V)$. Legyen A szűrők egy halmaza, ekkor a V-beli csúcsok $\phi(A, V)$ darab üzenetet kapnak. Így a cél az, hogy meghatározzuk azt a k elemű A halmazt, ahová a szűrőket majd elhelyezzük, úgy hogy a különbség maximális legyen $\phi(0, V)$ és $\phi(A, V)$ között.

`1. probléma (Filter Placement-FP):` Adott egy irányított c-gráf $G(V, E)$ és egy egész k szám. Adjuk meg a gráf csúcsainak azt az A részhalmazát , amely maximálisan k elemből áll, és amelyre maximális az $F(A) := \phi(0V) - \phi(A, V)$ függvény.

Az $F(A)$ függvény mindig pozitív és monoton, mivel egy szűrő elhelyezése csupán csökkenteni tudja az adatok és másolataik számát. Ebből az is következik, hogy F korlátos: $F(0) = 0 \leq F() \leq F(V)$. Továbbá az F függvény szubmoduláris, mivel minden $X \subset Y \subset V$ és $v \notin Y$ esetén, fennáll az
$F(X \cup \{v\}) - F(x) \geq F(Y \cup \{v\}) - F(Y)$ összefüggés. Az 1. probléma defi-

níciójában van egy k, korlátozott változó, ami a szűrők számát adja meg. A következőkben bemutatjuk, hogy ha k nem korlátos, akkor a megoldás triviális. Továbbiakban jelölje n a G-ben lévő csúcsok számát.

1. Lemma: Legyen $G(V, E)$ egy irányított c-gráf. Megtalálni a minimális méretű A halmazt, amelyre $F(A) = F(V)O(|E|)$ ideig tart.

Bizonyítás: Konstruáljuk meg A-t a következő módon: $A = \{v \in V | d_{in}(v) > 1$ és $d_{out}(v) > 0\}$. Ezeket a csúcsokat választva szűrőknek biztosak lehetünk benne, hogy minden élen csupán egy példánya fog egy adatnak áthaladni, hiszen ha nem szűrő egy csúcs, akkor vagy csupán egy él mutat rá, vagy nincs kimenő éle. Továbbá az A halmaz minimális is, hiszen, hogyha egy elemet elveszünk belőle, akkor már találkozni fogunk nemkívánatos másolatokkal is. A lemma ellenére az FP NP teljes probléma adott méretű k esetén.

1. Állítás: Az FP probléma egy tetszőleges c-gráfon NP-teljes.
A bizonyítás megtalálható a cikkben.

4. Szűrő elhelyező algoritmusok

A következő szakaszban algoritmusok lesznek bemutatva a FP probléma megoldására fákon, DAG-okon, illetve tetszőleges irányított gráfokon.

4.1. FP probléma fákon

Amíg az FP probléma NP-teljes tetszőleges gráfok esetén, addig polinomiális időben megoldható dinamikus programozással c-fákon. Egy $G(V, E)$ gráfot kommunikációs fának (c-fa) neveznek, hogyha azonkívül, hogy G c-gráf, fa is ha elhagyjuk a forrás csúcsot. A dinamikus programozás rekurziója minden csúcs gyermekein van végrehajtva. A bemenő c-fából egy bináris fát alakítanak ki, mivel így számítógéppel sokkal jobban kezelhető. Az átalakíás menete a következő: minden $v \in V$ csúcsra, hogyha maximum két kimenő éle van ne csináljunk semmit. Ellenkező esetben vegyünk egy $v_1, v_2, \ldots v_r$ sorrendjét a csúcs gyermekeinek, ahol r a kimenő csúcsok száma. Legyen $v1v$ bal gyereke. Csináljunk egy új u_1 csúcsot, és legyen az v jobb gyereke. A többi gyermek legyen u_1 gyermeke. Ismételjük ezeket a lépéseket amíg u_{r-1}-nek már csak két gyermeke marad v_{r-1} és v_r. Az így létrejövő bináris fa legyen G'-nek. Vegyünk észre azt, hogy G'-ben sokkal több csúcs van mint G-ben, továbbá ha a maximális kifok G-ben h, akkor G' $(h-1)$-el lesz magasabb mint G.
Az algoritmust G'-n fogják végrehajtani: Legyen $OPT(v, i, A)$ a függvény, amely

megtalálja az optimális szűrők halmazát a v gyökerű részfában, ahol $|A| \leq i \leq k$. Majd minden $i = 0...k$-ra ki tudják számolni az $OPT(v, i, A)$-t:
$OPT(v, i, A) = max\{$
$max_{j=0...i}\{OPT(v_l, j, A + OPT(v_r, i - j, A)\}$,
$max_{j=0...i-1}\{OPT(v_l, j, A \cup v) + OPT(v_r, i - 1 - j, A \cup \{v\})\}\}$.

A fenti összefüggésben a v_l és a v_r a v csúcs bal, illetve jobb gyermekét jelöli. A rekurzió első része azt az esetet nézi, amikor nem választjuk szűrőnek a v csúcsot, ekkor ugyanúgy i darab filtert lehet még elhelyezni összesen a a v_l és v_r gyökerű részfákban. A rekurzió második része az az eset, amikor v-t szűrőnek választjuk. Ekkor $i-1$ darab filtert helyezhetünk el a v_r és v_l gyökerű részfákban. Az optimális megoldást az egész fára az $OPT(r, k, A)$-val kaphatjuk, ahol r a fa gyökere, k a maximálisan használható szűrők száma és A kezdetben üres. A fenti rekurzió nem garantálja, hogy olyan csúcsot választunk ki szűrőként, ami nincs benne G-ben. Ezért a rekurzió második részét elhagyjuk ott, ahol a v nincs benne G-ben. Legyen Δ a maximális kifok G-ben. Ekkor a faépítés $O(n\Delta)$ ideig tart. A rekurziót $O(k)$-szor kell lefuttatni minden egyes csúcsra. Egyszeri futtatása a rekurziónak $O(k)$ számítást igényel. G'-ben $O(nlog(\Delta))$ csúcs van, amiből adódik, hogy a futási idő $O(n\Delta + k^2 nlog(\Delta))$.

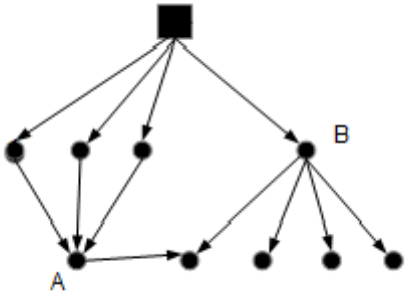
4.2. A Filter Placement probléma DAG-okra

Most vegyük a körmentes és irányított c-gráfokat (DAG). Annak ellenére, hogy úgy tűnik, hogy a DAG-ok egyszerűbbek a tetszőleges gráfoknál, már ezeken is NP-teljes az FP probléma.

2. `Állítás:` Az FP probléma NP-teljes, abban az esetben, ha a $G(V, E)$ c-gráf egy DAG.

További részében oylan polinom idejű algoritmusokat láthatunk, amik megoldják az FP problémát és hatékonyaknak bizonyultak a méréseik során.
Először egy naiv megözeltést használnak, $Greedy\_1$ (G_1). Vegyünk egy $v \in V$ csúcsot, v adatokat fog kapni a bejövő élein, és minden kimenő élén az adat egy másolatát fogja továbbítani. Most tudunk mondani egy alsó becslést azon másolatok darabszámára, amelyeket v továbbküld gyermekeinek: $m(v) = d_{in}(v) * d_{out}(v)$. A $Greedy\_1$ minden csúcsra kiszámolja $m(v)$-t, és kiválasztja azt a k darab csúcsot szűrőnek, amelyeknek a legnagyobbak az $m()$ értékeik. Az $m()$ érték kiszámításához általában $O(|E|)$ idő szükséges (függ a gráf tárolási módjától). A k darab legnagyobb $m()$ érték megtalálásához $O(kn)$ időre van szükségünk, így a $Greedy\_1$ algoritmus futási ideje $O(kn + |E|)$.

`Algoritmus Greedy_1:`
1: `Input`: DAG $G(V, E)$, k pozitív egész szám.
2: Output: A szűrők $A \subseteq V$ halmaza és $|A| \leq k$.
3: `while` V.VanmégElem() `do`
4: v = V.következőElem()
5: M $\leftarrow$ (v, d_{in}(v) $\times$ d_{out}(v))
6: A $\leftarrow$ {v : A k darab legnagyobb $M(v)$ érték}
7: `return` A



2. ábra: $k = 1$-re a $Greedy\_1$ a B csúcsot választja ki filternek, amíg az A csúcs lenne az optimális

Noha a $Greedy\_1$ egyszerű és hatékony, számos gráf esetén nem megfelfelő mint a 2. ábra is mutatja. Szűrők nélkül összesen 14 fogadott adat van a gráfban. $Greedy\_1$ a B csúcsot választaná ki szűrőnek, hiszen $m(B) = 1 * 4$ a legnagyobb $m()$ érték a gráfban, hiszen $m(A) = 3 * 1$. Viszont ha B-be teszünk egy szűrőt, azzal nem csökkentettük a redundanciát, míg ha az A csúcs lenne a szűrő, akkor 12-re csökkenne az összes fogadott adat mennyisége. Az előbbi példa fényében a $Greedy\_All$ (G_All) algoritmust javasolják, ami egy javított kiterjesztése $Greedy\_1$-nek. Az ötlet a $Greedy\_All$ algoritmushoz az, hogy számoljuk ki minden v csúcsra, hogy hány másolat keletkezik egy i adatból a v csúcs hatására. Az algoritmus ezután mohón kiválasztja a legnagyobb ilyen értékkel rendelkező csúcsot és oda tesz egy szűrőt. Először figyejük meg egy i csomag terjedés az s forrásból. Ahhoz, hogy $i - t$ megkapja egy v csúcs, i-nek egy irányított úton kell végighaladnia s-től v-ig. Tehát annyi másolatot fog kapni v i-ből ahány irányított út van s-ből v-be. Jelöljük az x csúcsból az y csúcsba futó különböző irányított utak számát $\#paths(x, y)$-nal. Tehát az s-ben generált i-ből kapott másolatok száma a v csúcsban $\#paths(s, v)$. Különbséget kell tenni azonban a különböző irányított utak száma, és egy i elemből érkezett másolatok számával. Az utóbbit $Prefix(v)$-vel jelöljük. Vegyük észre, hogy most $\#paths(s, v) = Prefix(v)$. Ezen felül, legyen $Suffix(v)$ az i-ből készült azon másolatoknak a számát, amelyek az v csúcs elérése után keletkeztek. Ez a mennyiség szintén az irányított

utakkal kapcsolatos. Most a $Suffix(v)$-t ki tudjuk számolni a v-ből induló összes irányított útként: $Suffix(v) = \sum_{u \in V} \#paths(v, u)$. Vegyük észre, hogy összesen $Prefix(v) * Suffix(v)$ darab a v csúcs által terjesztett i másolatoknak a száma. Vizsgálva egy szűrő elhelyezésének a hatását a v csúcsban, észrevehetjük, hogyha v szűrő, akkor v az i adat egy másolatát fogja csupán továbbítani. Másképpen ha a v csúcs filter, akkor $Prefix(v) = 1$. Így az összes, a v csúcs által generált redundánsan készült másolat i-ből kifejezhető a következő módon: $I(v) = (Prefix(v) - 1) * Suffix(v)$. Az $I(v)$ számot v hatásának nevezik. Ez a hatás kifejezhető a következőképpen is: $F(v) = \phi(0, V) - \phi(\{v\}, V) = I(v)$. Ez az elgondolás szolgál a *Greedy_All* algoritmus alapjaként: az algoritmus először kiválasztja a legnagyobb hatással bíró csúcsot. Kiválasztja azt szűrőnek, majd újraszámolja az $I(v)$ értékeket minden csúcsra. Ezután a *Greedy_All* megint kiválasztja a legnagyobb hatással bíró csúcsot. Az algoritmus ezeket a lépéseket k-szor ismétli meg. Vegyük észre, hogy amíg a *Greedy_1* algoritmus csupán egy csúcs környezetét veszi figyelembe, addig a *Greedy_All* az egész gráfon dolgozik, így futásideje is több lesz.

2. `Algoritmus` *Greedy_All*
`Input:` DAG $G(V, E)$ és k pozitív egész szám.
`Output:` A szűrők A $\subseteq V$ halmaza és $|A| \leq k$.
1: keressük meg a csúcsok egy δ topológikus sorrendjét.
2: `for` i = 1 . . . k `do`
3: `for` j = 1 . . . n `do`
4: számít $I(v_j)$
5: A $\leftarrow$ maximum $v \in V$ $I(v)$
6: `return` A

4.2.1. A *Greedy_All* algoritmus implementálása:

A hatás kiszámításához ki kell számolni a $Prefix()$ és $Suffix()$ értékekeket minden egye csúcsra. A $Prefix(v)$-re úgy is gondolhatunk, hogy egy adott i elemből hány másolatot kap a v csomópont. Mivel i másolatait v szülein keresztül kapjuk meg, ezért könnyen belátható, hogy egy csúcs $Prefix()$ értéke a szülei $Prefix()$ érétkeinek az összege. Minden csúcsra ki tudjuk ezt az értéket számolni rekurzívan, úgy hogy először a szülők $Prefix()$ értékeit számoljuk ki. Ehhez veszik a csúcsok egy δ topológikus sorrendjét. Ebből következik, hogy egy csúcsot megelőznek az ősei a rendezésben. Ha a G gráf csúcsait bejárjuk e rendezés szerint, akkor a $Prefix()$ értékek kiszámíthatóak a következő rekurzív formulával:

$$Prefix(v) = \sum_{x \in \pi_V} Prefix(x)(1)$$

Emlékezzünk, hogy a $Prefix(v)$ kiszámítható $\#paths(s, v)$-ként is, ezért az (1)-es formulával megegyezik:

$$Prefix(v) = \sum_{x \in \pi_V} \#paths(s, x) (2)$$

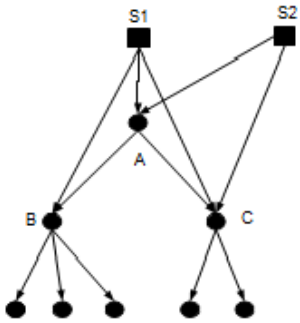
Mint tudjuk $Suffix(v)$ megegyezik a v-ből induló irányított utak számával a G gráfban. Ez az érték kiszámítható az (1) számítása közbeni rekurzió során egy lista segítségével. Minden v csúcs esetében legyen $plist_v$ egy lista. Legyen x a v csúcs egy őse, és a listában tároljuk el az x-ből v-be vezető irányított utakat. Tehát $plist_v[x] = paths(x, v)$. Vegyük észre, hogy v egy x őse esetén $plist_v[x]$-et ki lehet számolni a szülők plist értékeinek az összegeként.

$$\forall x \in V\colon\ plist_v\,[x] = \sum_{p \in \pi_V} plist_p\,[x]\ (3)$$

A $Suffix(v)$ kiszámításához összegezni kell v-ből induló utakat.

$$Suffix(v) = \#paths(v, .) = \sum_{x \in V} plist_x\ [x]\ (4)$$

A megvalósításhoz minden csúcs listájának tartalmaznia kell önmagát 1-es értékkel: $plist_v\,[v] = 1$. Mostmár tudjuk, hogyan kell kiszámolni a $Suffix$ és a $Prefix$ értékeket, hogyha nincsen szűrő a gráfban. Azt tudjuk, hogyha a $v*$ csúcsot szűrőnek választjuk, akkor úgy is tekinthetnénk mintha csupán egy út lenne s-ből $v*$-ba. Ezt egyszerűen el lehet érni a $plist_v\,[v*] = 1$, $plist_x\,[x] = 0$ változtatással (x minden nem $v*$ csúcs). Figyeljünk, hogy ez a változtatás a $Suffix$ és $Prefix$ értékeket is megváltoztathatja. A *Greedy_All* optimális $k = 1$ esetben és nagyobb k esetén is nagyon jól működik az algoritmus. Észre lehet venni, hogy vannak olyan, a való életben is megtalálható gráfok, amelyeken a *Greedy_All* algoritmus megtalálja a megfelelő szűrőket. Azonban vannak esetek, amikor nem megfelően választ, például vegyük a harmadik ábrát. Amikor nincsenek szűrők elhelyezve, akkor az összes fogadott adat $\phi(q, V) = 26$. $K = 1$ esetében $I(A) = 7, I(B) = 6, I(C) = 6$, tehát az algoritmus az A csúcsba fog egy szűrőt tenni. Az értékek frissítése után $I(B|A) = 3, I(C|A) = 4$. Az algoritmus a C csúcsot fogja választani. Most a $\phi(\{A, C\}, V) = 15$. Viszont ha a B, illetve a C csúcs lenne szűrő, akkor $\phi(\{B, C\}, V) = 14$ lenne, ami kisebb mint 15. A *Greedy_All* algoritmus futásideje $O(k * n * log(n))$.



3. ábra: $k = 2$-re a $Greedy\_All$ az $\{A, C\}$ halmazt választja szűrőknek a $\{B, C\}$ halmaz helyett.

4.2.2. A $Greedy\_All$ algoritmus gyorsítása

Mivel nagy adatszerkezetek esetén, a $Greedy\_All$ algoritumus meglehetősen lassú, ezért közlik két lehetséges gyorsítását az algoritmusnak. Az első algoritmus a $Greedy\_Max$ (G_Max), kiszámoljan minden csúcsra a hatást, hasonlóképpen mint a $Greddy\_All$. Ezután a $Greedy\_Max$ algoritmus kiválasztja a k legnagyobb értékkel rendelkező csúcsot mint szűrőt, a hatások újraszámolása nélkül. A méréseik azt mutatják, hogy a $Greedy\_Max$ nagyon hasonló csúcsokat választ ki mint a $Greedy\_All$ algoritmus. A futásideje $O(n|E|)$, hiszen csak egyszer kell kiszámítani a hatást. Egy másik heurisztika a $Greedy\_L$ (G_L). Ebben az esetben egy egyszerűsített hatást számolnak ki minden csomópont esetén: $I'(v) = Prefix(v) * d_{out}(v)$, ami megadja, hogy v hány adatot propagál a közvetlen gyermekeinek. Ezekután a $Greedy\_L$ kiválasztja a k legnagyob ilyen hatással rendelkező csúcsot filternek. $I'()$ kiszámításához ki kell számítani a $Prefix()$ értékeket és tudni kell minden csúcs kifokát. Mindkét feladat G egyszeri bejárásával kivitelezhető. I'-t minden iteráció során frissíteni kell, így kapva $O(k * |E|)$-s futásidőt.
Mindhárom heurisztikát használó algoritmus ($Greedy\_1, Greedy\_Max, Greedy\_L$) gyorsabb mint a $Greedy\_All$ algoritmus. Ezt a gyorsulást úgy érték el a cikk írói, hogy kevesebb információ segítségével számítják ki a szűrőket. Mindegyik heurisztika álltalában más szűrőket választ ki, így a teljesítményük is más különböző gráfokon. Vegyünk például egy nagy be- illetve kifokkal rendelkező csúcsot, ekkor a G_1-ben nagy értéket fogunk hozzárendelni. Ugyanakkor G_1 nem veszi vigyelembe a csomópont helyét a gráfban. G_Max ugyankkor a teljes hatást kiszámítja, így sokkal jobban meg tudjuk becsülni vele a különböző csúcsok hatását. Ugyanakkor az azonos utakon lévő csúcsok esetében nem veszi észre a

korrelációt, így egy úton több szűrőt kiválasztva csökkenhet a szűrők hatásfoka. $Greedy\_L$ ezeket próbálja meg csökkenti a két módszer kombinálásával. Azonban G_L álltalában a forrástól messzi csúcsokat fog választani filternek, hiszen a forrástól távolodva a $Prefix()$ érték exponenciálisan nő.

3. `Algoritmus` $Greedy\_L$
`Input:` DAG $G(V, E)$ és k pozitív egész szám.
`Output:` A szűrők $A \subseteq V$ halmaza és $|A| \leq k$.
1: $A = \{\}$
2: `for` $i = 1...k$ `do`
3: számít $Prefix()$
4: A $\leftarrow$ maximum $v \in VPrefix(v)$
5: `return` A

4.3. Szűrőelhelyezés tetszőleges gráfokon

Ez egy NP-teljes probléma, megoldásához kiválasztják a gráf egy DAG tulajdonságú részgráfját. Így a feladat megoldható az előző részben bemutatott algoritmusok segítségével. Legyen $G'(V, E')$ egy tetszőleges c-gráf. Vegyük a csúcsok egy δ topologiku sorrendjét. Egy (v, u) élet előremutatónak nevezünk, hogyha $\delta(v) < \delta(u)$, ellenkező esetben hátrafelé mutató él. Egy általános módszer egy DAG kiválasztására a következő: Legyen δ a csúcsok egy topologikus sorrendje, F a δ által meghatározott előre mutató élek halmaza, B a maradék élek halmaza. Ha $|F| < |B|$, akkor válasszuk, a DAG $G(V, F)$-et, ellenkező esetben a $G(V, B)$-t. Ez az algoritmus azonban nem garantálja, hogy összefüggő gráfot kapunk, ezért a cikk írói kifejlesztették a saját algoritmusukat($Acyclic$), hogy kiválasszanak egy összefüggő DAG-et. Az $Acyclic$ algoritmus 2 lépésre osztható. Tegyük fel, hogy csupán egy s forrás van G-ben (ellenkező esetben egy új gráf készítése egy s' forrás csúccsal és onnan vezetnek irányított élek a régi forrás csúcsokba). Először egy mélységi keresést hajtanak végre s-ből. Minden él, amit a bejárás során érintenek hozzáadódik G-hez. Ezután a maradék élek közül E'-ben, azok kerülnek bele G-be, amelyek nem hoznak létre kört. Az így készített részgráf maximális, hiszen nem adható hozzá további olyan E'-beli él, hogy ne keletkezne kör. A mélységi bejárás során épített fát F-fel jelölik. Ha egy elem a forrásból eljut a v csúcsba, akkor benne lesz ebben a fában is a v csúcs. A mélységi keresés során érintett élek G egy feszítőfáját eredményezik, így G összefüggő lesz. Az algoritmus második részében egy él akkor kerül G-be hogyha az nem eredményez kört. Erre egy megoldás hogy hozzáadjuk az e élt G-hez és megnézzük, hogy körmentes-e, ha nem akkor kivesszük az e élet G-ből. Ehelyett egy döntési mechanizmust használnak a T-beli csúcsok segítségével. A csúcsok

sorrendjét a mélységi keresés során felfedezési időnek nevezik, és $\delta()$-val jelölik. Egy csúcsot csomópontnak nevezünk ha több mint egy gyermeke van T-ben. A mélységi keresés miatt nem lehet E'-ben olyan előre mutató él ($\delta()$-val számolva), ami nincs benne T-ben sem. Egy visszafelé mutató (u, v) élet hozzá lehet adni a E-hez, hogyha nincsen irányított út v-ből u-ba. Ezek azok az esetek amikor v és u a fa különböző ágain helyezkednek el. Így van egy w csomópont, melyre egy ilyen irányított út található: (w, w_{u_1}) ... (w_{u_r}, u) és $(w, w_{v_1})...(w_{v_l}, v)$ T-ben, és w_{u_1} és w_{v_l} különböző csomópontok. Annak érdekében, hogy eldöntsük mely w-t kell megtartani, minden u csúcs esetén tárolni kell egy $sign(u)$-t, ami $\{(w, w_{u_1})\}$ párok listája. Itt a párok első eleme (w), a csomópontok az $(s \rightarrow u)$ úton. Továbbá $\delta(w_{u_1})$ kisebb vagy egyenlő mint $\delta(u)$. Továbbá minden w-ben kezdődő ág esetén, minden csúcs felfedezési ideje nagyobb vagy egyenlő mint $\delta(w_{u_1})$. Mikor egy (u, v) élet hozzá akarunk adni, megnézzünk a $sign(u)$ értéket és a $sign(v)$-t. Megkeresük azt a legnagyobb $\delta(w)$ érétkű w-t, amelyre $(w, w_{u_1}) \in sign(u)$ és $(w, w_1) \in sign(v)$. u és v akkor és csak akkor található a fa különböző ágain, ha $\delta(v) < \delta(w_{u_1}) \leq \delta(u)$. Az algoritumus futási ideje $O(n2 * logn)$.

`4. algoritmus maximális DAG kiválasztása.`
`Input:` irányított $G'(V, E')$ gráf
`Output:` DAG $G(V, E)$
1: mélységi bejárás s kezdéssel
2: $E \leftarrow T$
3: a $sign()$-ek kiszámítása
4: `for` $(u, v) \in E'$ `do`
5: `if` $\delta(v) < \delta(w_{u_1}) \leq \delta(u)$ `do`
6: $E \leftarrow (u, v)$

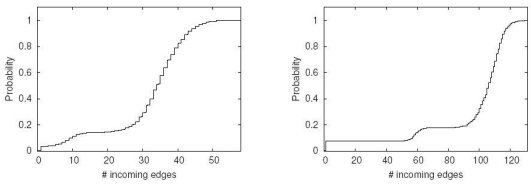
5. Kísérleti értékelések

A cikk írói bemutattak egy módszert, amivel össze lehet hasonlítani a különböző szűrő elhelyező algoritmusokat mind szintetikus mind való életből vett adatok esetén. Bemutatják a különböző algoritmusok hatékonyságát és a futási idejüket is. Teljesítmény mérése: Ahhoz, hogy a különböző algoritmusokat össze lehessen haonlítani bevezetik a Filter Ratio-t $(FR) : FR(A) = F(A)/F(V)$, ahol A azon csomópontok halmaza, ahová a szűrőket helyezzük el, V pedig az egész gráf csúcshalmaza. Kiindulási algoritmusok: Az algoritmusaikat összehasonítják néhány véletlenen alapuló algoritmussal is, hogy a hatékonyságukat objektíven mérni tudják. *Random_k* (Rand_K): Ez az algoritmus kiválaszt véletlenszerűen k darab csúcsot V-ből, ezek lesznek a szűrők. *Random_Independent* (Rand_I): Minden

csúcs függetlenül a másiktól k/n valószínűséggel lesz szűrő. *Random_Weighted* (Rand_W): Minden v csúcs $w(v) \times n$ valószínűséggel lesz szűrő, ahol $w(v) = \sum_{u \in C_v} 1/(d_{in}(u)$ és $C_v = \{u \in V | (u, v) \in E\}$. C_v a v csúcs gyermekei. Észre lehet venni, hogy a véletlenített algoritmusokban nem garantált, hogy pontosan k darab szűrő lesz. A véletlenített algoritmusok eredményeit 25 futás átlagaként számolják.

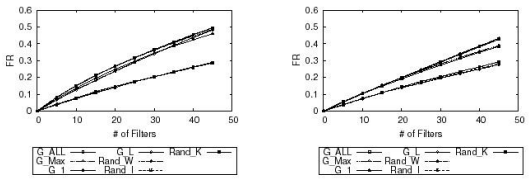
5.1. Eredmények szintetikus adatok használata esetén

Először 10 szintbe rendelték a csúcsokat véletlenszerűen úgy, hogy szintenként várhatóan 100 csúcs kerüljön. Ezután irányított éleket generáltak egy i szintű v csúcsból minden $j > i$ szintű u csúcsba $p(u, v) = x/(y_j - i)$ valószínűséggel. Az x és az y választásától függ a gráf sűrűsége. $X/y = 1/4$ esetén 1026 csúcs és 32427 él állt elő, illetve $x/y = 3/4$ esetén 1069 csúcs és 101226 él állt elő. A 4. ábrán láthatjuk a CDF-t bejövő élekre. A kép hasonló lenne kimené élek esetén is.



(a) $x = \frac{1}{4}$ (b) $x = \frac{3}{4}$

4. ábra: A CDF bejövő élek esetén mesterséges gráfokra



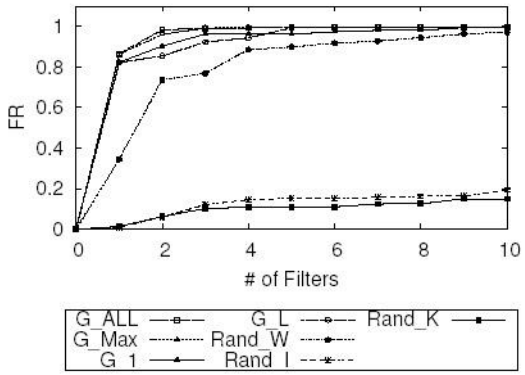
(a) $x = \frac{1}{4}$ (b) $x = \frac{3}{4}$

5. ábra: FR mesterséges gráfok esetén

Az 5. ábrán azt láthatjuk, hogy az FR függvény értéke folyamatosan növekszik, ahogyan növeljük a szűrők számát. Ez azt mutatja, hogy a szűrőként választott csomópontok nagyjából azonos méretű különböző útjait fedik le a gráfnak.

5.2. Eredmények valós adatok esetén

A *Quote* [8] tartalmazza a média oldalakatól kezdve a nagy hírforgalmozókon át a személyes blogokig minden link hálózatot időbélyegekkel. Mivel ennek a gráfnak több 100000 van, ezért csupán egy részgráfját használják a cikk írói, amit G_Phrase-nek neveznek. Erre a gráfra futtatták az *Acyclic* algoritmust, így DAG-et kaptak ami 932 csúcsból és 2703 élből áll. A gráf csúcsainak körülbelül a felének a kifoka egy, továbbá vannak olyan csúcsok is, amiknek nagy a ki-, illetve a befokuk, ezek potenciális szűrők.

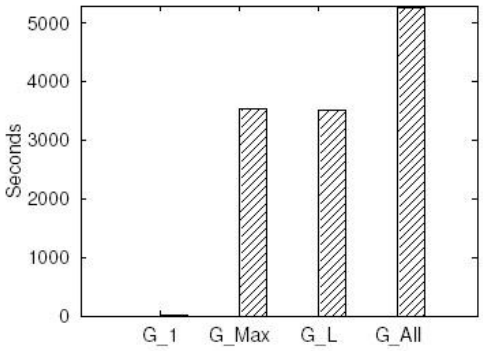


6. ábra: FR a G-Phrase esetén

A 6. ábrán láthatjuk az eredményeket. Mint várható volt a G_ALL algoritmus végzett a legjobban. G_MAX k = 2-re más csúcsot választ ki, de ezután ugyanazokat választotta mint a G_ALL, így ez is ez is majdnem olyan jó teljesítményt nyújt mint a G_ALL. A két heurisztikával támogatott algoritmus (G_1, G_L) is csupán kevéssel maradt le a G_MAX-tól. A teljesen randomizált algoritmusok lényegesen rosszabb eredménnyel zártak mint a cikk írói által közölt algoritmusok.
A $Random\_Weighted$ viszont egész jó eredményekkel zárt, hiszen annak a csúcsnak van nagyobb súlya, amelyiknek több a gyermeke. A cikkben még méréseket végeztek a Twitter [1], illetve az APS [6] [7] adatszerkezetkre is. Itt is az emelhető ki, hogy a $Greedy\_all$ algoritmus végzett a legjobb helyen, míg a teljesen véletlenített algoritmusok hatékonysága igen csekély.

5.3. Futásidők

4GHz AMD Opteron, 256GB of RAM, 64-bit Linux CentOS. Az algoritmusokat Python nyelven implementálták, illetve az előző mondatban leírt konfiguráción futtatták. A mérési eredményeket az alábbi ábrán láthatjuk.



7. ábra: futási idők tíz filter,
Twitter adatszerkezet esetén

Mint látható a *Greedy_1* algoritmus a leggyorsabb, kevesebb mint egy perces eredményével. A leglassabb a *Greedy_all* a 83 perces futásidejével. Végül a *Greedy_L*, illetve a *Greedy_Max* hasonló egy óra környéki futásidőkkel végeztek. Összefoglalva az előzőeket, a *Greedy_1*, *Greedy_Max*, *Greedy_L* algoritmusok sokkal hatékonyabb futásidőben mint a *Greedy_All* algoritmus, így nagyobb adatok esetén ajánlottabb lehet ezeket az algoritmusokat használni. Az előbbi megfigyelést azzal a ténnyel párosítva, hogy az összes a cikkben bemutatott algoritmus hatékonyan megoldja a FP problémát, arra a következtetésre jutunk, hogy a gyakorlatban használni lehet a heurisztikával támogatott algoritmusokat is.

6. irodalomjegyzék

[1]H. Kwak, C. Lee, H. Park, and S. Moon. What isTwitter, a social network or a news media? In ACMWWW '10, pages 591–600, 2010

[2]J. Aspnes, K. Chang, and A. Yampolskiy. Inoculationstrategies for victims of viruses and the sum-of-squarespartition problem. In YALEU/DCS/TR-1295, 2004.

[3]K.-P. Chan and A. W. chee Fu. Efficient time series matching by wavelets. In ICDE, pages 126–133, 1999.

[4]S. ching S. Cheung and A. Zakhor. Efficient video similarity measurement with video signature. IEEE TCSVT, 13:59–74, 2003.

[5]A. Natsev, R. Rastogi, and K. Shim. Walrus: A similarity retrieval algorithm for image databases. In IEEE TKDE, pages 395–406, 1999.

[6]https://publish.aps.org/datasets

[7]O. Rader, W. Gudat, C. Carbone, E. Vescovo,S. Blügel, R. Klasges, W. Eberhardt, M. Wuttig,J. Redinger, and F. J. Himpsel. Electronic structure of two-dimensional magnetic alloys:c(2x2) mn on cu(100) and ni(100). Phys. Rev. B, 55(8):5404–5415, 1997.

[8]J. Leskovec, L. Backstrom, and J. Kleinberg. Meme-tracking and the dynamics of the news cycle. In 15th ACM SIGKDD, pages 497–506, 2009.

[9]Dora Erdös,Vatche Ishakian, Andrei Lapets, Evimaria Terzi, Azer Bestavros Filter-Placement Problem and its Application to Minimizing Information Multiplicity