Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Lauter Kinga Csilla

RÉSZGRÁF-IZOMORFIA PROBLÉMA VIZSGÁLATA

KONZULENS

Dr. Szegletes Luca

BUDAPEST, 2020

Tartalomjegyzék

Összefoglaló

Dolgozatomban a részgráf-izomorfia problémáját mutatom be. A gráfok és a részgráf keresés több területen is hasznosítható, például bioinformatikában, szociális hálók esetén az adatok feldolgozásánál, gyógyszerkutatásnál, illetve áramkörök tervezésénél.

A gráfok csúcspontokból és csúcspontokat összekötő élekből állnak. A részgráf-izomorfia azt jelenti, hogy G_1 és G_2 gráfokat nézve van-e olyan G_2' részgráfja G_2-nek, amelyik izomorf G_1-gyel, azaz létezik egy-egyértelmű megfeleltetés (bijekció) G_2' és G_1 csúcsai és élei között. Ez a probléma NP-teljes, vagyis nem ismerünk polinomiális idejű algoritmust a megoldására, így fontos a lehető legjobb keresési módszer megtalálása, főleg nagy gráfok esetén.

Dolgozatomban Julian R. Ullmann 1976-os algoritmusát, Luigi P. Cordella és munkatársai 2004-es VF2 algoritmusát, Jüttner Alpár és Madarasi Péter 2018-as VF2++ algoritmusát, valamint Myoungji Han és munkatársai 2019-es, dinamikus programozáson alapuló DAF algoritmusát mutatom be és hasonlítom össze különböző gráf adatbázisokon, ezeken megvizsgálom működésüket, teljesítményüket.

Abstract

In my paper I am going to examine the subgraph isomorphism problem. Graphs and subgraph matching are playing an important role in many fields, e.g. bioinformatics, social networks, chemical research, electric circuits.

Graphs consist of vertices and edges. Given two graphs, G_1 and G_2, subgraph isomorphism means whether there is a G_2' subgraph of G_2 that is isomorphic to G_1. In other words there is a bijection between the vertices and edges of G_1 and G_2'. This problem is NP-complete, therefore we don't know of any polinomial algorithms to solve it. It is important to find the best possible method especially in case of large graphs.

In my paper I am going to present and compare the performance and other factors of the following algorithms: Julian R. Ullmann's algorithm (1976), the VF2 algorithm by Luigi P. Cordella et al (2004), the VF2++ algorithm by Alpár Jüttner and Péter Madarasi (2018) and the dynamic programming based DAF algorithm by Myoungji Han et al (2019).

1. Bevezetés

A mérnöki világban a gráfoknak számos alkalmazási területe van, kezdve a különböző hálózatok vizsgálatától a bioinformatikán keresztül a szerkezeti kémiáig. A szerkezeti kémia területén például az egyes atomok megfeleltethetők a gráf csúcsainak, míg a kötések a gráf éleinek. Molekulák vizsgálatában azonnal felmerülhet, hogy mikor nevezhetünk két molekulát hasonlónak. Ilyenkor az a feltételezésünk, hogyha két molekula szerkezete hasonló, akkor hasonlóak lesznek a tulajdonságaik is.

Munkámban a szerkezetei hasonlóságot vizsgálom, vagyis a gráfizomorfiát, ezt kiterjesztem a részgráf-izomorfia problémakörre, amikor is nemcsak két gráf hasonlóságát vizsgálom, hanem hogy az egyik gráf tartalmazza-e a másik gráfot.

A részgráf-izomorfia NP-teljes, ami azt jelenti, hogy nincs polinom időben lefutó algoritmus a megoldására. Ez akkor jelent igazán nagy problémát, amikor nagy, többmillió csúcsú gráfon szeretnénk ezeket az algoritmusokat lefuttatni úgy, hogy akár ezer részgráfot szeretnénk megvizsgálni. Dolgozatomnak ez a motivációja, hogy megvizsgáljam ezt a problémakört, a szakirodalomban ismert algoritmusokat.

1.1 Dolgozatom felépítése

Dolgozatomat az alapfogalmak bevezetésével kezdem (gráf, részgráf-izomorfia). A harmadik fejezetben bemutatom a szakirodalomban ismert algoritmusokat, majd a negyedik fejezetben kitérek az implementációjukra. Az ötödik fejezetben a bemutatott, megvalósított algoritmusokat ismert adathalmazokon, nagy gráfokon tesztelem, összehasonlítom az eredményeiket. Végül összefoglalom a dolgozatomat és ismertetem a továbbfejlesztési lehetőségeket.

1.2 Jelölések

A részgráf kereső algoritmusokban a következő jelöléseket használom: a két gráf q $(query)$ és G $(data)$, ahol $|q| \leq |G|$ tehát a cél a q-val izomorf részgráfok keresése G-ben. $V(q) = u_{1,2,\dots|V(q)|}$ és $V(G) = v_{1,2,\dots|V(G)|}$, azaz q-hoz tartozó csúcsokat u_i-vel, G-hez tartozó csúcsokat pedig v_j-vel jelölöm. A (rész)tartalmazás jelölése M, amennyiben sorrend felállítása szükséges, azt O betűvel jelölöm.

2. Alapfogalmak bevezetése

Az algoritmusok megértéséhez a következő fogalmak, tételek szükségesek [1] [2]:

2.1 Definíció (Gráf)

Egy gráf egy rendezett pár $G = (V, E)$ ahol V egy nem-üres halmaz, elemei a gráf csúcsai (pontjai), E pedig a V halmazból képezhető párok egy halmaza, elemei a gráf élei. Ha egy G gráfról beszélünk, akkor $V(G)$-vel jelöljük a gráf csúcsainak halmazát, $E(G)$-vel pedig a gráf éleinek halmazát. A gráf rendje, vagyis a csúcspontok száma $|V(G)|$, míg a gráf éleinek száma $|E(G)|$.

Egy $e = \{v_1, v_2\} \in E$ él esetén a v_1 és v_2 csúcsokat az él végpontjainak nevezzük.

Két pont, v_1 és v_2 szomszédosak, ha $\{v_1, v_2\} \in E$, tehát ha van közöttük él. Egy e él irányított, ha az élben a v_1, v_2 csúcsok sorrendje kötött, és irányítatlan, ha nem. Irányított éleket tartalmazó gráfot irányított gráfnak, irányítatlan éleket tartalmazó gráfot pedig irányítatlan gráfnak hívjuk.

Egy v pontra illeszkedő élek számát a pont fokszámának nevezzük, jelölése: $d(v)$ vagy $deg(v)$.

Azt a G gráfot, amelyben nincsen $\{v_i, v_i\}$ él (hurokél) vagy azonos csúcsok között két vagy több él (többszörös élek), egyszerű gráfnak nevezzük.

Egy G gráfot címkézett gráfnak hívunk, ha a csúcsaihoz és/vagy éleihez címkéket rendelünk. Ekkor v csúcs esetén a címkét $lab(v)$-vel, e él esetén pedig $lab(e)$-vel jelöljük. Ha a csúcsokhoz és élekhez nem rendelünk címkéket, akkor a gráfot címkézetlen gráfnak hívjuk.

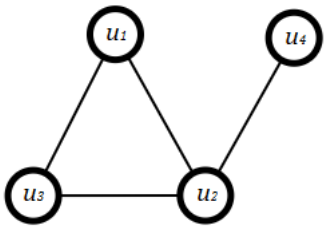
Egy G gráf ritka, ha az élek száma sokkal kevesebb az élek maximális számánál, tehát c $|V(q)|^2$-nél.

2.2 Gráfok ábrázolása

2.2.1 Szomszédsági mátrix

Egy G gráfot többféleképpen ábrázolhatunk. Egyik leggyakoribb mód a szomszédsági (adjacencia) mátrix (jele: $A(G)$), ahol G csúcsai a mátrix sorai és oszlopaiként

szerepelnek, egy adott sor és oszlop által kijelölt elem értéke azt jelenti, hogy a sornak és az oszlopnak megfelelő csúcsok között van él a gráfban. Egyszerű gráf esetén a mátrixban 1 és 0 értékek lehetnek, attól függően, hogy a két csúcs között van-e él vagy nincs.
Például: adott a következő G gráf



1. ábra: G gráf

Ekkor G szomszédsági mátrixa:

$$A(G) = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Egyszerű gráf esetén a szomszédsági mátrix főátlójában 0 szerepel, mert egyik csúcs sem rendelkezik hurokéllel. A mátrixban például az első sor és második oszlop által kijelölt elem 1-es értéke az $\{u_1, u_2\}$ élnek felel meg. Irányítatlan gráfok esetén a szomszédsági mátrix a főátlóra szimmetrikus, mert az $\{u_i, u_j\}$ él megfelel az $\{u_j, u_i\}$ élnek, ha az élnek nincs iránya.

2.2.2 Éllista

A gráfok egy másik gyakori leírási módszere az éllista. Ekkor listában tároljuk azokat az (u_i, u_j) párokat, amelyek között létezik él a gráfban. Ritka gráfok esetén jobb lehet ez az ábrázolás, hiszen a szomszédossági mátrixban sok 0 szerepelne, amelyet sokszor felesleges tárolni.
Példa: A fenti G gráf éllistája: $L = (u_1, u_2), (u_1, u_3), (u_2, u_3), (u_2, u_4)$

2.3 Definíció (Izomorf gráfok)

A $G = (V, E)$ és a $G' = (V', E')$ gráfok izomorfak, ha van olyan egy–egy értelmű megfeleltetés – bijekció – a csúcsok, V és V' között, hogy G-ben pontosan akkor szomszédos két pont, ha G'-ben a nekik megfelelő pontok szomszédosak, és szomszédos pontpárok esetén ugyanannyi él fut közöttük.

2.4 Definíció (Részgráf)

A $G' = (V', E')$ gráf a $G = (V, E)$ gráf részgráfja, ha $V' \subseteq V$, $E' \subseteq E$, valamint egy pont és egy él pontosan akkor illeszkedik egymásra G'-ben, ha G-ben is illeszkedők.

2.4.1 Definíció (Részgráf-izomorfia I.)

Az előző két definícióból megkaphatjuk a részgráf-izomorfia (vagy részgráf keresés) definícióját:
Adott $G = (V_G, E_G)$ és $H = (V_H, E_H)$ gráfok. G részgráf-izomorf H gráffal, ha létezik G-nek olyan $G' = (V'_G, E'_G)$ részgráfja ($V'_G \subseteq V_G$, $E'_G \subseteq E_G$), hogy G' és H izomorfak, tehát létezik olyan bijekció V'_G és V_H, között, hogy G'-ben pontosan akkor szomszédos két pont, ha H-ban a nekik megfelelő pontok szomszédosak, és szomszédos pontpárok esetén ugyanannyi él fut közöttük.

2.4.2 Definíció (Részgráf-izomorfia II.)

Legyen $G = (V, E)$, $H = (V', E')$ gráfok. G részgráf-izomorf H-val, ha G-nek van olyan részgráfja $G_0 = (V_0, E_0) \mid V_0 \subseteq V, E_0 \subseteq E$, hogy létezik olyan bijekció $f: V_0 \rightarrow V'$, hogy $\{v_1, v_2\} \subseteq E0 \leftrightarrow \{f(v_1), f(v_2)\} \subseteq E'$.

2.5 NP-teljesség kérdése

Az algoritmusok bonyolultság szempontjából több osztályra oszthatók. Azok a kérdések, amelyek polinomiális időben megválaszolhatók, a P (polinomial (time)) bonyolultsági osztályba tartoznak. Azonban vannak olyan kérdések, amelyek megválaszolására nem ismert polinomiális idejű algoritmus, de egy kapott megoldás ellenőrzésére igen. Ezek a

NP (nondeterministic polinomial (time)) bonyolultsági osztályba tartozó kérdések: azok a kérdések tartoznak ide, amelyek polinomiális időben bizonyíthatók.
A P-NP probléma 1971. óta a számítógép-tudomány egyik legfontosabb megoldatlan kérdése. A feltételezés szerint P ≠ NP, ez azt jelenti, hogy vannak olyan kérdések, amelyekre egy válasz ellenőrzése polinomiális idő alatt lehetséges, a kérdés megoldása kapott válasz nélkül azonban nem. Ha P = NP, akkor minden probléma, amely ellenőrizhető polinomiális időben, polinomiális időben megoldható is, ezt a legtöbb kutató nem tartja valószínűnek [3].

2.5.1 Definíció (NP-teljes probléma)

Egy problémát NP-teljesnek hívunk, ha az NP bonyolultsági osztályba tartozik, és bármely más NP-beli probléma polinomiális időben visszavezethető rá.
Formálisan:
Egy P döntési probléma NP-teljes, ha:

1. $P \in NP$
2. $\forall\, P' \in NP - re: P' \leq_{polinomial} P$

ahol $\leq_{polinomial}$ a polinomiális időben visszavezethetőséget jelenti („polinomszor nehezebb”). Ha egy problémánál csak a második feltétel teljesül, akkor azt NP-nehéznek hívjuk.
Tehát ha egy NP-teljes problémára sikerülne polinomiális idejű algoritmust találni, akkor a visszavezethetőség miatt az összes NP-teljes problémára használható lenne ez a megoldás.

2.6 Tétel (Részgráf-izmorfia probléma NP-teljes)

A részgráf-izomorfia probléma NP-teljes, azaz jelenlegi tudásunk szerint nem oldható meg polinomiális idő alatt. Ennek bizonyítása a Hamilton-kör probléma részgráf-izomorfia problémára való visszavezetéssel történik.

2.6.1 Definíció (Hamilton-kör)

Egy G gráfban Hamilton-körnek nevezzük azt a H kört, amelyik a G gráf minden csúcspontját pontosan egyszer tartalmazza.

A Hamilton-kör probléma, azaz eldönteni, hogy egy G gráfban van-e Hamilton-kör, NP-teljes. [1]

Ahhoz, hogy megmutassuk, hogy egy probléma, esetünkben a részgráf-izomorfia probléma NP-teljes, a definícióban megadott két feltétel teljesülésének bizonyítása szükséges.

1. $P \in NP$

A részgráf-izomorfia problémánál belátható, hogy NP-beli, hiszen, ha q és G gráf esetén rendelkezésre áll egy tanú, amely megmutatja G' G részgráfját, valamint q és G' csúcspontjai közötti M párokat, akkor ellenőrizhető, hogy ez megfelel-e a részgráf-izomorfia feltételeinek.

2. $\forall\, P' \in NP - re: P' \leq_{polinomial} P$

Az NP-nehézség belátásához visszavezetjük a Hamilton-kör problémát az részgráf-izomorfia problémára. Adott G gráf, meg szeretnénk tudni, hogy van-e benne Hamilton-kör. Ha a q gráfot úgy adjuk meg, hogy egy olyan kör, amely csúcsainak száma megegyezik G csúcsainak számával, akkor látható, hogy visszavezettük a Hamilton-kör problémát a részgráf-izomorfia problémára: ha G-ben létezik Hamilton-kör, az akkor, és csak akkor lehet, ha G-nek létezik q-val izomorf részgráfja, mivel q egy Hamilton-körnek felel meg G-ben. G és q csúcsainak száma ekkor megegyezik, mivel a Hamilton-körben a gráf minden csúcsa szerepel, G éleinek száma több, mint q éleinek száma, mert Hamilton-körben minden csúcs fokszáma pontosan kettő. Ha találunk egy olyan részgráfot G-ben, amely izomorf q-val, akkor meg tudjuk mondani, hogy G-ben van-e Hamilton-kör.

A részgráf-izomorfia probléma tehát NP-teljes.

Az NP-teljes problémákra nem ismerünk polinomiális idejű algoritmust, így a részgráf-izomorfia problémára sem.

3. Algoritmusok bemutatása

Ebben a fejezetben bemutatom a részgráf kereső algoritmusokat. Elsőként a legtriviálisabb, nyers erő (brute force) algoritmussal kezdem, amely a leglassabb, így viszonyításként szolgálhat a többi algoritmushoz, amelyek gyorsabban megtalálják a q gráffal izomorf részgráfokat G gráfban. Újabb algoritmusok közül bemutatom Ullmann algoritmusát, a VF2 algoritmust, a VF2++ algoritmust, valamint a DAF algoritmust.

3.1 Nyers erő (brute force) algoritmus

Elsőként megemlítem a nyers erő (brute force) módszert, amellyel a részgráf-izomorfia probléma megoldható, amikor G-ben minden lehetséges részgráfhoz megnézzük, hogy izomorf-e q-val. Ennek menete: minden $u_i \in V(q)$ csúcshoz megkeressük azokat a $v_j \in V(G)$ csúcsokat, amelyek lehetséges jelöltjei u_i-nek: v_j fokszáma legalább akkora, mint u_i fokszáma, és címkézett gráfok esetén u_i és v_j címkéje megegyezik. Az így kapott jelölthalmazokból mindegyik u_i csúcs esetén egyet választunk, így egy keresőfát építünk fel, majd a végén ellenőrizzük, hogy a fában kapott utak közül melyek felelnek meg a részgráf-izomorfia definíciójában lévő feltételeknek. Ennek időigénye azonban nagy gráfok esetén hatalmas lehet.

A következő algoritmusok jobb, gyorsabb módszereket mutatnak be.

3.2 Ullmann algoritmus

Az első megvizsgált új algoritmus Julian Richard Ullmann algoritmusa, amelynek első változata 1976-ban [4] jelent meg, egy második, kiegészített változata pedig 2011-ben [5]. Az algoritmus egy keresőfát épít fel, amelynek segítségével G gráfban megtalálja az összes q gráffal izomorf részgráfot [6].

Az algoritmus első lépésben minden $u_i \in V(q)$ csúcshoz megkeresi a $v_j \in V(G)$ jelölteket, azokat a v_j csúcsokat, amelyek fokszáma legalább akkora, mint u_i fokszáma, valamint címkézett gráfok esetén a $v_j$csúcshoz rendelt címke megegyezik u_i címkéjével.

$$C(u_i) = \{\, v_j \mid v_j \in G, \deg(u_i) \leq deg(v_j) \ \wedge\ lab(u_i) = lab(v_j)\}$$

Ezután q csúcsainak egy O sorrendjét kell vennünk, majd egy keresési fát építünk fel a sorrendnek és az adott csúcshoz tartozó jelölteknek megfelelően. Ezt a sorrendet úgy kapjuk meg, hogy az $u_i \in V(q)$ csúcsokat fokszámaik szerint csökkenő sorrendbe tesszük. A nulladik szint az üres halmaz, az első szinten a sorrendben az első u_i csúcs és minden jelöltje található. A második szinten a sorrend szerinti második u_i csúcs, valamint jelöltjei, a többi szinten hasonlóan, így az (u_i, v_j) párok fát alkotnak. A bővítés során az ún. $filtering$ és $refinement$ szabályokat használjuk, amellyel biztosítjuk az egy-egyértelműséget, valamint azt, hogy egy u_i csúcshoz a csúcs szomszédjai szempontjából is megfelelő v_j csúcsot találjunk.

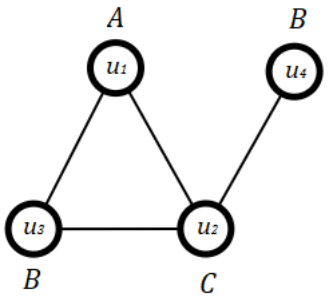
$Szint(i) = (O[i], v_j \in C(O[i]))$, ahol $O[i]$ a sorrendben az i-edik elemet jelenti (egytől indulva).

Egy (u_i, v_j) pár létrehozása után a sorrendben u_i utáni csúcsok jelöltjei közül kivesszük v_j-t, amennyiben szerepel ott, ez biztosítja, hogy a tartalmazás egy-egyértelmű legyen, tehát két különböző u_i, u_k csúcshoz ne kerüljön ugyanaz a v_j pár. Az algoritmusban ez a folyamat a $szűrés$ $(filtering)$ nevet viseli.

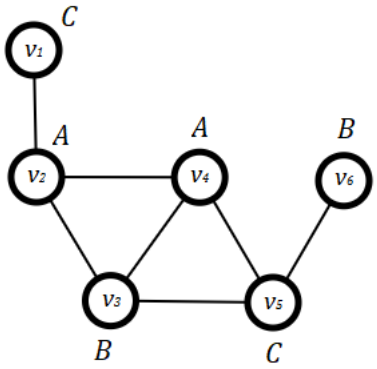
Ezután a $finomítás$ $(refinement)$ rész következik, ahol ellenőrizzük u_i szomszédjainak jelöltjeit. Az aktuális u_i csúcs minden szabad n szomszédjának c jelöltjére megnézzük, hogy G-ben létezik-e $\{v_j, c\}$ él, ha nem, akkor c-t kivesszük az adott ágon n jelöltjei közül. Itt tehát azt nézzük, hogy c szomszédos-e v_j-vel, ez azért szükséges, mert u_i szomszédos n-nel, és v_j u_i már megtalált párja, ha nem szomszédosak, akkor az (n, c) pár az adott ágon nem vezetne jó megoldáshoz, így felesleges lenne azon haladni.

Ezek a változtatások azt eredményezhetik, hogy valamely u_i csúcsnak nem marad jó jelöltje. Ekkor, tehát ha a fa bővítése során olyan ághoz jutunk, ahol a sorrendben a következő csúcsok között van olyan, amelynek jelölt halmaza üres, akkor biztos, hogy azon az ágon nem fogunk tartalmazáshoz jutni, ezért nem kell tovább haladni. Ha a fa mélysége akkora, mint q csúcsainak száma, akkor minden $u_i \in V(q)$ csúcshoz találtunk $v_j \in V(G)$ párt, így G-ben megtaláltunk egy q-val izomorf részgráfot. Ekkor visszaléphetünk az előző szintre, ahonnan folytatjuk az esetleges többi tartalmazás keresését.

Példa: Adott az alábbi két gráf (q és G)



2. ábra: q gráf



3. ábra: G gráf

$C(u_1) = \{v_2, v_4\}$, mert u_1-hez az A címke tartozik, fokszáma 2, G-ben ennek a két feltételnek v_2 és v_4 felel meg.

$C(u_2) = \{v_5\}$, mert u_2 címkéje C, fokszáma 3, G-ben csak v_5-re igaz a feltétel, v_1 esetén a címke megegyezik, de fokszáma kisebb, mint u_2 fokszáma.

$C(u_3) = \{v_3\}$, mert u_3 címkéje B, fokszáma 2, G-ben B címkével v_3 és v_6 rendelkezik, de v_6 fokszáma csak 1.

$C(u_4) = \{v_3, v_6\}$, mert u_4 címkéje B, fokszáma pedig 1.

A csúcsok sorrendje a fokszám szerinti csökkenő sorrend: $O = \{u_2, u_1, u_3, u_4\}$

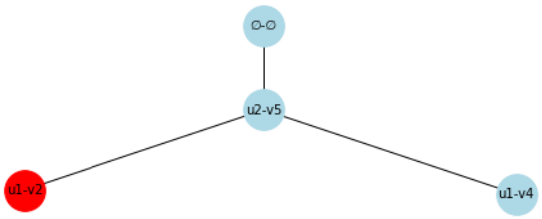
Ekkor a fa nulladik szintje az üres halmaz, tehát a $\{\emptyset, \emptyset\}$ pár.

Az első csúcs sorrendben az u_2, jelöltje v_5, így a fa első szintjére (u_2, v_5) pár fog kerülni. A $szűrés$ során nem kell változtatni a jelölteken, ugyanis más csúcsnak nem jelöltje v_5. A $finomítás$ résznél megnézzük, hogy u_2 szomszédjai, azaz u_1, u_3 és u_4

jelöltjei közül van-e olyan, amely nem szomszédos v_5-tel G-ben. u_1 jelöltjei közül v_2 nem szomszédos v_5-tel, így eltávolítjuk. A másik két szomszéd, u_3 és u_4 minden jelöltje szomszédos v_5-tel, így jelöltjeikből nem kell elvenni. Egyik csúcs jelöltjeinek halmaza sem üres, így tovább léphetünk a következő szintre.

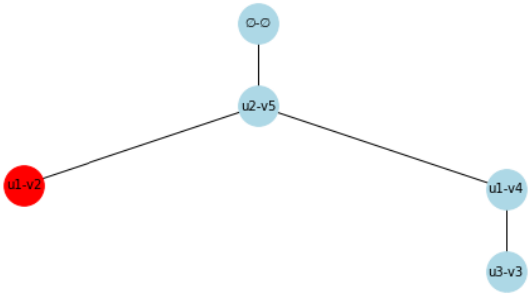
4. ábra: A keresőfa 1. szintje

A második csúcs a sorrendben az u_1, jelöltje csak v_4 maradt, így a fa második szintjére az (u_1, v_4) pár fog kerülni. A $szűrés$ résznél nem változtatunk a többi jelölten, ugyanis v_4 nem szerepel más csúcs jelöltjei között. A $finomítás$ résznél megnézzük u_1 szabad szomszédjainak jelöltjeit: u_3 jelöltje v_3, amely szomszédos v_4-gyel. Egyik csúcs jelöltjeinek száma sem nulla, ezért léphetünk tovább a következő szintre.



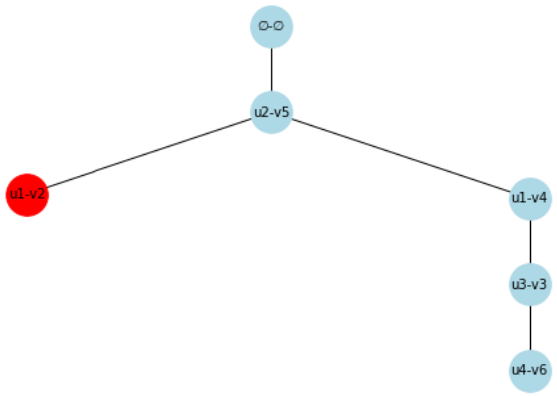
5. ábra: A keresőfa 2. szintje (pirossal szerepel u_1 a finomítás miatt eltávolított v_2 jelöltje)

A sorrendben a harmadik csúcs az u_3, amelynek jelöltje v_3, így felvesszük az (u_3, v_3) párt a fa harmadik szintjébe. A $szűrés$ során u_4 jelöltjei közül eltávolítjuk v_3-at. $Finomítás$ nem szükséges, mert u_3-nak nincs olyan szomszédja, amely még nem szerepel a fában. Nincs olyan csúcs, amelynek nincs jelöltje, ezért léphetünk a következő szintre.



6. ábra: A keresőfa 3. szintje

Az utolsó csúcs az u_4, amelynek v_6 jelöltjét felvéve a fába, eljutunk ahhoz, hogy a szint száma megegyezik q csúcsainak számával, így megtaláltuk G-ben egy q-val izomorf részgráfot. Ezután visszalépések következnek, eljutunk a nulladik szintig, ahol véget ér az algoritmus, mert nincs több jelölt.



7. ábra: A keresőfa 4. szintje

Az Ullmann algoritmus által felépített keresőfa tehát megmutatta, hogy q gráf G gráf egyetlen részgráfjával izomorf, a csúcspárok a következők: $(u_1, v_4), (u_2, v_5), (u_3, v_3), (u_4, v_6)$.

3.3 VF2 algoritmus

A VF2 algoritmust 2004-ben Luigi Pietro Cordella, Pasquale Foggia, Carlo Sansone és Mario Vento publikálták [7], az algoritmus különösen nagy gráfok esetén hatékony. Irányított, címkézett gráfok szerepelnek az algoritmus leírásában.

Az algoritmus egy résztartalmazást bővít folyamatosan, addig, amíg minden $u \in V(q)$ csúcsnak lesz párja. Egy résztartalmazást $M(s)$ jelöl, ahol s az aktuális állapot. A teljes tartalmazás, azaz a részgráf-izomorfizmus jele M. $M(s)$ tehát q és G egy-egy részgráfját mutatja meg, az $M(s)$-ben szereplő q-beli csúcsokat $M_q(s)$-sel, G-beli csúcsokat pedig $M_G(s)$-sel jelöljük.

$$M_q(s) = \{ u \mid u \in V(q) \wedge \exists\, v \in V(G) : (u, v) \in M(s) \}$$

$$M_G(s) = \{ v \mid v \in V(G) \wedge \exists\, u \in V(q) : (u, v) \in M(s) \}$$

Az s állapotból s' állapotba lépés annak feleltethető meg, hogy s-be felveszünk egy új (u', v') párt. Egy adott állapothoz tartozó jelöltpárok halmazát $P(s)$-sel jelöljük.

Az algoritmus kezdőállapota az s_0, ekkor a résztartalmazás üres: $M(s_0) = \emptyset$. Ez egy helyes, az izomorfia szabályainak megfelelő résztartalmazás. A résztartalmazás bővítéséhez minden $M(s)$ résztartalmazáshoz meghatározzuk a jelöltpárok $P(s)$ halmazát. Egy tartalmazáshoz többféleképpen eljuthatunk, ennek kiküszöbölésére minden bővítés előtt azon $u \in V(q)$ csúcsok között, amelyek $P(s)$-ben benne vannak, definiálunk egy teljes sorrend relációt (total order relation, jele: $\prec$). Mivel a tartalmazás meghatározásánál mindegy, hogy a csúcspárokat a résztartalmazásba milyen sorrendben vettük fel, ezért az algoritmus csak a teljes sorrend reláció első $u \in V(q)$ csúcsot és $v \in V(G)$ párjait nézi, tehát minden olyan $(u', v') \in P(s)$ párt figyelmen kívül hagy, amelyre igaz, hogy a $P(s)$-ben van olyan $u \in V(q)$, hogy $u \prec u'$.

Miután eltávolítottuk a sorrend szerint nem megfelelő párokat, megkaptuk a jó jelöltpárok halmazát. Minden $p \in P(s)$ párhoz a bővítés előtt ellenőrzi, hogy teljesülnek-e a megfelelő feltételek: a résztartalmazás egy-egyértelmű, valamint a később bemutatott $megvalósíthatósági\ szabályok$ $(feasibility\ rules)$ teljesülnek-e. Ha ezek a feltételek teljesülnek, akkor a $p = (u, v)$ párt felvesszük s állapotba, így s' állapot kapunk. Ekkor kiszámoljuk az új jelöltpárokat. Így a résztartalmazások egy keresőfát alkotnak. Ezen a fán az algoritmus mélységi bejárás (DFS, Depth-first search) [8] sorrendjében halad végig. Ha egy részen a jelöltek elfogynak, akkor az algoritmus visszalép egy szintet. Ha a nulladik szintről lépnénk vissza, tehát s_0 állapothoz nincs több jelölt, akkor az algoritmus leáll.

3.3.1 Jelöltek kiszámítása

Egy adott állapotnál a jelölt párok $P(s)$ halmazához először meghatározzuk a következőket:

$T_q^{out}(s)$ és $T_G^{out}(s)$ azon $u \in V(q)$ és $v \in V(G)$ csúcsok halmaza, amelyek még nem szerepelnek $M(s)$-ben, és amelyekbe futnak élek $M_q(s)$, illetve $M_G(s)$-beli csúcsból, tehát olyan $u' \in V(q)$, illetve $v' \in V(G)$ csúcsból, amely szerepel $M(s)$-ben.

$$T_q^{out}(s) = \{u \mid u \in V(q) \wedge u \notin M_q(s) \wedge \exists u' \in V(q): u' \in M_q(s) \wedge (u', u) \in E(q)\}$$

$$T_G^{out}(s) = \{v \mid v \in V(G) \wedge v \notin M_G(s) \wedge \exists v' \in V(G): v' \in M_G(s) \wedge (v', v) \in E(G)\}$$

$T_q^{in}(s)$ és $T_G^{in}(s)$ azon $u \in V(q)$ és $v \in V(G)$ csúcsok halmaza, amelyek még nem szerepelnek $M(s)$-ben, és amelyekből futnak élek $M_q(s)$, illetve $M_G(s)$-beli csúcsa, tehát olyan $u' \in V(q)$, illetve $v' \in V(G)$ csúcsba, amely szerepel $M(s)$-ben.

$$T_q^{in}(s) = \{u \mid u \in V(q) \wedge u \notin M_q(s) \wedge \exists u' \in V(q): u' \in M_q(s) \wedge (u, u') \in E(q)\}$$

$$T_G^{in}(s) = \{v \mid v \in V(G) \wedge v \notin M_G(s) \wedge \exists v' \in V(G): v' \in M_G(s) \wedge (v, v') \in E(G)\}$$

A $P(s)$ halmaz eleme ekkor minden (u, v) pár lesz, ahol $u \in T_q^{out}(s)$ és $v \in T_G^{out}(s)$. Ha ezek közül valamelyik üres, akkor $T_q^{in}(s)$ és $T_G^{in}(s)$ csúcsait nézzük. Nem összefüggő gráfok esetén vagy az első csúcs vizsgálatakor előfordulhat azonban, hogy ezek a halmazok is üresek, ekkor $P(s)$ halmazba minden $M(s)$ résztartalmazásban még nem szereplő $u \in V(q)$ és $v \in V(G)$ pár kerül.

3.3.2 Megvalósíthatósági szabályok (feasibility rules)

Sok olyan állapot létezhet, amelyek nem vezetnek részgráf izomorfizmushoz, fontos, hogy ezeket az állapotokat minél hamarabb azonosítsuk, hogy ne kelljen feleslegesen bővíteni azokat. Szükséges, hogy minden állapot konzisztens legyen, a generált állapotok száma csökkenthető úgy is, hogy egy, illetve két lépéssel előretekintünk, és megnézzük, hogy akkor is létezne-e olyan csúcspár, amely az akkori állapothoz felvéve konzisztens állapothoz vezetne. Ezek a szabályok összefoglalóan a *megvalósíthatósági szabályok* (*feasibility rules*) nevet viselik. Két nagy részre bonthatók, a szintaktikai megvalósíthatóság a gráf struktúráját nézi, míg a szemantikai az attribútumokat, tehát a címkéket, amennyiben léteznek. Ötféle szintaktikai megvalósíthatósági szabály létezik, ezek irányított gráf esetén ellenőrzik az adott csúcspontok szüleit, gyermekeit.

A szabályok bemutatásához először a következő halmazok definiálása szükséges:

$T_q(s)$ a $T_q^{out}(s)$ és $T_q^{in}(s)$ halmazok uniója, $T_G(s)$ a $T_G^{out}(s)$ és $T_G^{in}(s)$ halmazoké.

$$T_q(s) = T_q^{out}(s) \cup T_q^{in}(s) \qquad T_G(s) = T_G^{out}(s) \cup T_G^{in}(s)$$

$\widetilde{N}_q(s) = V(q) - M_q(s) - T_q(s)$, tehát azon $V(q)$ csúcsok halmaza, amely nem szerepel a résztartalmazásban, és nem szülője vagy gyermeke egy résztartalmazásbeli csúcsnak sem. $\widetilde{N}_G(s)$ hasonlóképpen definiálható.

G gráfban v csúcs szüleit a következőképp jelöljük: $Pred(G, v)$, gyermekeit pedig $Succ(G, v)$ halmazként. Hasonlóképpen q gráfra és u csúcsra: szülők jelölése $Pred(q, u)$, gyermekek jelölése $Succ(q, u)$.

A jelöltek meghatározása után szükséges eltávolítani a sorrend szerint nem megfelelő párokat, így megkapjuk a jó jelöltpárok halmazát. Minden $p \in P(s)$ párra ellenőrizzük a következő öt megvalósíthatósági szabályt, ezek megmutatják, hogy a résztartalmazás az új csúcspárral bővítve továbbra is vezethet-e tartalmazáshoz:

Legyen s az adott állapot, (u, v) pár pedig az ellenőrizendő pár.

R_{pred}: u minden résztartalmazásban szereplő u_p szülőjének párja v szülője. v minden résztartalmazásban szereplő v_p szülőjének párja u szülője.

$$\forall\, u_p \in M_q(s) \cap Pred(q,u)\ \exists\, v_p \in Pred(G,v): (u_p, v_p) \in M(s)$$

$$\forall\, v_p \in M_G(s) \cap Pred(G,v)\ \exists\, u_p \in Pred(q,u): (u_p, v_p) \in M(s)$$

R_{succ}: u minden résztartalmazásban szereplő u_c gyermekének párja v gyermeke. v minden résztartalmazásban szereplő v_c gyermekének párja u gyermeke.

$$\forall\, u_p \in M_q(s) \cap Succ(q,u)\ \exists\, v_p \in Succ(G,v): (u_p, v_p) \in M(s)$$

$$\forall\, v_p \in M_G(s) \cap Succ(G,v)\ \exists\, u_p \in Succ(q,u): (u_p, v_p) \in M(s)$$

R_{in}: v-nek van legalább annyi $T_G^{in}(s)$-ben szereplő szülője, mint u-nak $T_q^{in}(s)$-ben szereplő szülője. v-nek van legalább annyi $T_G^{in}(s)$-ben szereplő gyereke, mint u-nak $T_q^{in}(s)$-ben szereplő gyereke.

$$\left|Pred(q,u) \cap T_q^{in}(s)\right| \leq \left|Pred(G,v) \cap T_G^{in}(s)\right|$$

$$\left|Succ(q,u) \cap T_q^{in}(s)\right| \leq \left|Succ(G,v) \cap T_G^{in}(s)\right|$$

R_{out}: v-nek van legalább annyi $T_G^{out}(s)$-ben szereplő szülője, mint u-nak $T_q^{out}(s)$-ben szereplő szülője. v-nek van legalább annyi $T_G^{out}(s)$-ben szereplő gyereke, mint u-nak $T_q^{out}(s)$-ben szereplő gyereke.

$$\left|Pred(q,u) \cap T_q^{out}(s)\right| \leq |Pred(G,v) \cap T_G^{out}(s)|$$

$$\left|Succ(q,u) \cap T_q^{out}(s)\right| \leq |Succ(G,v) \cap T_G^{out}(s)|$$

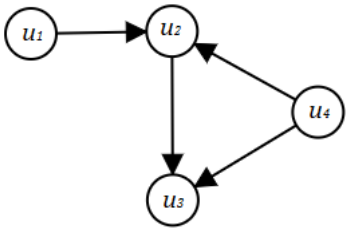
R_{new}: v-nek van legalább annyi $\widetilde{N}_G(s)$-ben szereplő szülője, mint u-nak $\widetilde{N}_q(s)$-ben szereplő szülője. v-nek van legalább annyi $\widetilde{N}_G(s)$ben szereplő gyereke, mint u-nak $\widetilde{N}_q(s)$-ben szereplő gyereke.

$$\left|Pred(q,u) \cap \widetilde{N}_q(s)\right| \leq \left|Pred(G,v) \cap \widetilde{N}_G(s)\right|$$

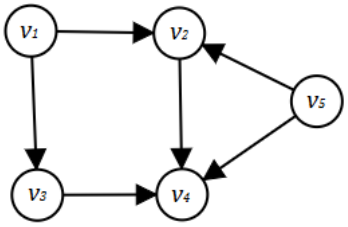
$$\left|Succ(q,u) \cap \widetilde{N}_q(s)\right| \leq \left|Succ(G,v) \cap \widetilde{N}_G(s)\right|$$

Az első két szabály az új $M(s')$ konzisztenciáját ellenőrzi $M(s)$-hez képest, a többi pedig a keresőfa vágásában segít. Ha a keresőfa egy szintjén minden lehetséges P(s) párjelöltet, és minden jó jelölthöz tartozó részfát bejártunk, vissza kell lépnünk egy szintet. Ha a nulladik szintről, s_0-ról lépnénk vissza, akkor az algoritmus leáll.

Példa: Adott az alábbi két gráf (q és G)



8. ábra: q gráf



9. ábra: G gráf

Ekkor a kezdő résztartalmazás: $M(s_0) = \emptyset$. $T_q^{in}(s_0) = \emptyset$, $T_q^{out}(s_0) = \emptyset$, $T_G^{in}(s_0) = \emptyset$, $T_G^{out}(s_0) = \emptyset$, $T_q(s_0) = \emptyset$, $T_G(s_0) = \emptyset$, $M_q(s_0) = \emptyset$, $M_G(s_0) = \emptyset$, $\widetilde{N}_q(s_0) = u_1, u_2, u_3, u_4$, $\widetilde{N}_G(s_0) = v_1, v_2, v_3, v_4, v_5$. Mivel $T_q^{out}(s_0)$, $T_G^{out}(s_0)$, $T_q^{in}(s_0)$ és $T_G^{in}(s_0)$ is üres, így minden (u_i, v_j): $u_i \in V(q)$ és $v_j \in V(G)$ csúcspár kerül $P(s_0)$-ba. A sorrend legyen a csúcsok számozási sorrendje, tehát minden olyan (u_i, v_j) párt kiveszünk, ahol $i \neq 1$. Az így megmaradt párok: $(u_1, v_1), (u_1, v_2), (u_1, v_3), (u_1, v_4), (u_1, v_5)$.

10. ábra: A kezdő résztartalmazás üres (s_0)

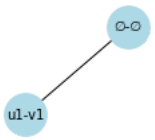
Egy pár résztartalmazásba felvétele előtt ellenőrizni kell a megvalósíthatósági szabályokat.

Nézzük s_0 állapotnál (u_1, v_1)-re:

Először u_1 és v_1 szüleit és gyermekeit határozzuk meg. $Pred(q, u_1) = \emptyset$, $Succ(q, u_1) = u_2, Pred(G, v_1) = \emptyset$, $Succ(G, v_1) = v_2, v_3$.

Most a megvalósíthatósági szabályok következnek, mivel $M(s_0)$ üres, ezért R_{pred} és R_{succ} teljesül. $T_q^{in}(s_0)$, $T_G^{in}(s_0)$ és $T_q^{out}(s_0)$, $T_G^{out}(s_0)$ halmazok üressége miatt az R_{out} és R_{in} szabályok is igazak lesznek. Az R_{new} szülőkre vonatkozó része teljesül, mert u_1-nek és v_1-nek nincsenek szülei q-ban, illetve G-ben. R_{new} gyerekekre vonatkozó részénél ellenőrizni kell, hogy v_1 $\widetilde{N}_G(s_0)$-beli gyermekeinek száma legalább akkora-e, mint u_1 $\widetilde{N}_q(s_0)$-beli gyermekeinek száma. $|u_2| \leq |v_2, v_3|$, így ez a feltétel is teljesül.

Minden megvalósíthatósági szabály igaz lett, felvehetjük az (u_1, v_1) párt s_0-ba, így kapjuk s_1 állapotot. A keresési fán mélységi bejárás szerint megyünk végig, ezért ezen az új ágon haladunk tovább.



11. ábra: s_0-ból az (u_1, v_1) pár felvételével kapjuk az s_1 állapotot

$M(s_1) = (u_1, v_1)$, $T_q^{in}(s_1) = \emptyset$, $T_q^{out}(s_1) = u_2$, $T_G^{in}(s_1) = \emptyset$, $T_G^{out}(s_1) = v_2, v_3$, $T_q(s_1) = u_2$, $T_G(s_1) = v_2, v_3$, $M_q(s_1) = u_1$, $M_G(s_1) = v_1$, $\widetilde{N}_q(s_1) = u_3, u_4$, $\widetilde{N}_G(s_1) = v_4, v_5$. $T_q^{out}(s_1)$ és $T_G^{out}(s_1)$ nem üres, így $P(s)$-be (u_2, v_2) és (u_2, v_3) kerül. Csak egyféle $u_i \in V(q)$ van, ezért most nem szükséges nézni a sorrendet. Először (u_2, v_2)-re ellenőrizzük a megvalósíthatósági szabályokat.

Szülők és gyermekek: $Pred(q, u_2) = u_1, u_4$, $Succ(q, u_2) = u_3$, $Pred(G, v_2) = v_1, v_5$, $Succ(G, v_2) = v_4$.

R_{pred}: u_2 résztartalmazásban szereplő szülője u_1, amelynek párja v_1, ami pedig v_2 szülője, így a feltétel teljesül

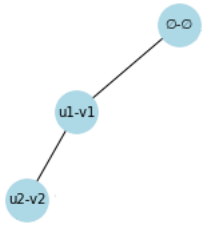
R_{succ}: se u_2, se v_2 gyermeke nem szerepel még a résztartalmazásban, azért ez a feltétel teljesül

R_{in}: $T_q^{in}(s_1)$ és $T_G^{in}(s_1)$ is üres, ezért a feltétel teljesül

R_{out}: u_2-nek nincs $T_q^{out}(s_1)$-ben szereplő szülője, v_2-nek sincsen $T_G^{out}(s_1)$-ben szereplő szülője: $|\emptyset| \leq |\emptyset|$, ugyanez igaz a u_2 és v_2 gyermekeire is

R_{new}: u_2 $\tilde{N}_q(s_1)$-beli szülője u_4, míg v_2 $\tilde{N}_G(s_1)$-beli szülője v_5, $|u_4| \leq |v_5|$, u_2 $\tilde{N}_q(s_1)$-beli gyermeke u_3, míg v_2 $\tilde{N}_G(s_1)$-beli gyermeke v_4, $|u_3| \leq |v_4|$, a feltétel teljesül.

Mind az öt szabály engedélyezi az új (u_2, v_2) pár felvételét s_1-be, így kapjuk a következő állapotot, s_2-t. Ezen az ágon haladunk tovább.



12. ábra: s_1-be felvesszük az (u_2, v_2) párt, így kapjuk s_2 állapotot

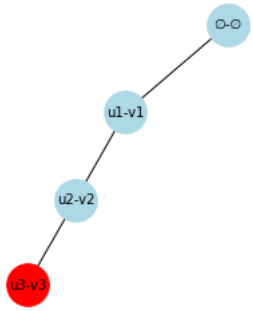
$M(s_2) = ((u_1, v_1), (u_2, v_2))$, $T_q^{in}(s_2) = u_4$, $T_q^{out}(s_2) = u_3$, $T_G^{in}(s_2) = v_5$, $T_G^{out}(s_2) = v_3, v_4$, $T_q(s_2) = u_3, u_4$, $T_G(s_2) = v_3, v_4, v_5$, $M_q(s_2) = u_1, u_2$, $M_G(s_2) = v_1, v_2$, $\tilde{N}_q(s_2) = \emptyset$, $\tilde{N}_G(s_2) = \emptyset$.

$T_q^{out}(s_2)$ és $T_G^{out}(s_2)$ nem üres, ezért $P(s_2)$-be (u_3, v_3) és (u_3, v_4) kerül. Csak egyféle q-beli csúcs van, azért most sem kell sorrendet alkalmaznunk. Először próbáljuk az (u_3, v_3) párral bővíteni a résztartalmazást:

Szülők és gyermekek kiszámítása: $Pred(q, u_3) = u_2, u_4$, $Succ(q, u_3) = \emptyset$, $Pred(G, v_3) = v_1$, $Succ(G, v_3) = v_4$.

R_{pred}: u_3 résztartalmazásban szereplő szülője u_2, melynek párja v_2, azonban ez v_3-nak nem szülője. v_3 résztartalmazásban szereplő szülője v_1, melynek párja u_1, azonban ez sem u_3 szülője. Így ez a feltétel nem teljesül, az s_2 résztartalmazás (u_3, v_3)

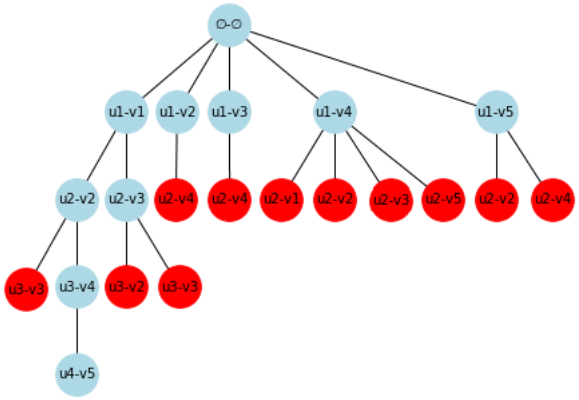
párral való bővítése nem vezetne tartalmazáshoz. Ezért visszalépünk a fában, s_2 másik jelöltpárjával próbálkozunk.



13. ábra: s_2 (u_3, v_3) párral történő bővítése nem vezetne tartalmazáshoz, ezért ezen az ágon nem haladunk tovább, P(s_2) másik párját ellenőrizzük

Most próbáljuk (u_3, v_3) párral bővíteni az $M(s_2)$ résztartalmazást. Ekkor minden feltétel teljesül, így felvehetjük az új párt.

A gondolatmenetet folytatva a többi jelöltre, a következő keresőfát kapjuk (ahol szükséges, az $u_i \in V(q)$ csúcsok sorrendje a számozásuk szerint növekvő sorrend):



14. ábra: VF2 algoritmus során előállt keresőfa

A fában piros színnel jelöltem azokat a párokat, amelyek a megvalósítási szabályokat nem teljesítik, emiatt nem megfelelőek. Így az algoritmus megtalálta G q-val izomorf részgráfját: (u_1, v_1), (u_2, v_2), (u_3, v_4), (u_4, v_5).

3.4 VF2++ algoritmus

A VF2++ algoritmust Jüttner Alpár és Madarasi Péter publikálta 2018-ban [9]. Az algoritmus a VF2 algoritmust egészíti ki, q csúcsait első lépésben sorrendbe teszi, majd ez alapján halad rajtuk végig, valamint a $v \in V(G)$ jelöltek kiválasztása is hatékonyabb

lett. Az algoritmus lényegesen gyorsabb, mint az előző algoritmusok. A cikkben irányítatlan, címkézett gráfok szerepelnek.

A VF2++ algoritmus első lépése a q csúcsainak sorrendbe rakása, a sorrend O lesz. Sok algoritmus csak a csúcsok fokszámait és a címkéket veszi figyelembe, azonban például a kémiában, biológiában gyakran ritka gráfok vannak, a csúcsoknak nem nagy a fokszámuk, és sok csúcsnak azonos, így nem hatékony sorrendet kapnánk. A VF2++ csúcssorrendje figyelembe veszi a fokszámot, a címke gyakoriságát, valamint az adott csúcs a résztartalmazásban már szereplő szomszédjainak számát.

Először kiválasztja a legritkább címkéjű és ezek közül a legnagyobb fokszámú csúcsot, az lesz a később felépítendő fa gyökere.

$$root = \{ r \mid r \in \arg max_{deg}(\arg min_{label}\,(V(q) \backslash O))\}$$

ahol $\arg min_{label}\,(V(q) \backslash O)$ azt jelenti, hogy az O sorrendben még nem szereplő $u \in V(q)$ csúcsok közül kiválasztjuk azokat, amely címkéjének O sorrendben előfordulásának számát kivonva G gráfban az adott címke előfordulásának számából, a legkisebb értéket kapjuk. Az $\arg max_{deg}$ pedig q gráfban az előbb megadott halmazból a legnagyobb fokszámú csúcsokat jelenti. Ha több ilyen van, akkor ezek közül egyet választunk a fa gyökerének.

Az algoritmus így meghatározott gyökértől kezdve szélességi bejárással (Breadth-first search, BFS) [8] felépít egy T fát, amelyben egyre mélyebbre megy.

$$T = bfs\_fa(gráf\colon q, gyökér\colon root)$$

A $0, 1, \dots, depth(T)$ szinten kiválasztja az adott szinten lévő csúcsokat, ezek lesznek a V_d halmaz csúcsai. Ezt a halmazt a következőképp dolgozza fel: kiválasztja azokat a csúcsokat, amelyeknek a legnagyobb a konnektivitása, azaz az O sorrendben már szereplő szomszédjainak száma. Ezek közül a legnagyobb fokszámúakat választja ki, majd a legritkább címkéjű csúcsot. Ha ezt az o csúcsot megtaláltuk, akkor a V_d-ből eltávolítjuk, és az O sorrend végére rakjuk.

$$o = \{ u \mid u \in \arg min_{label}(\arg max_{deg}(\arg max_{connectivity}(V_d)))\}$$

ahol $\arg min_{label}$ és $\arg max_{deg}$jelentése megegyezik a korábban leírtakkal a megfelelő halmazon, $\arg max_{connectivity}(V_d)))$ pedig azon $u \in V(q)$ csúcsok halmaza, amelyeknek a legtöbb O sorrendben szereplő szomszédja van.

A T fa egy szintjének feldolgozása addig tart, amíg a szinten van még olyan csúcs, amely nem szerepel az O sorrendben.

Ha a T fa minden szintjét feldolgoztuk, és az O sorrend még nem teljes, akkor visszatérünk az első lépéshez, és a maradék csúcsok közül választjuk ki a legritkább címkéjű, legnagyobb fokszámú csúcsot, majd a választott csúcstól kezdve építjük fel az új T fát. Miután minden csúcs bekerült a sorrendbe, kész vagyunk.

Az algoritmus további változtatása a VF2 algoritmushoz képest egy adott $u \in V(q)$ csúcs jelöltjeinek kiválasztása. Itt u csúcshoz elég keresnünk egy u_n szomszédot, amely már szerepel a résztartalmazásban. Mivel u_n szerepel a M résztartalmazásban, már van párja, legyen ez v_n. Így elég megnéznünk v_n szabad v_c szomszédjait, minden v_c csúcsnál ellenőrizni kell, hogy u többi résztartalmazásbeli szomszédjának párjával szomszédos-e.

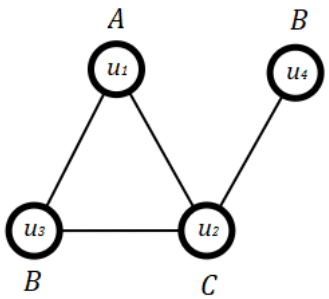
$v \in G$ *jelölt* $u \in q - hoz, ha$:

$$\forall u' \in q: \{u, u'\} \in E(q) \wedge u' \in M_q \Rightarrow \{v, M[u']\} \in E(G)$$

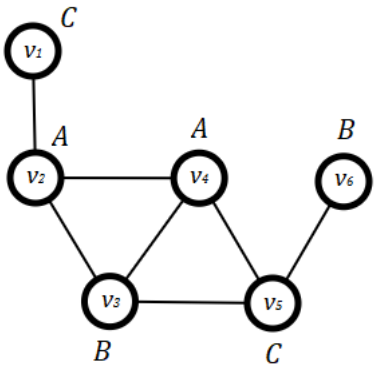
ahol M a résztartalmazás, M_q a résztartalmazásban szereplő $u \in V(q)$ csúcsok, $M[u']$ pedig u' párja a résztartalmazásban. Az első csúcs vizsgálatakor, amikor a résztartalmazás még üres, nem találunk szomszédos résztartalmazásban szereplő csúcsot, ekkor u csúcs jelöltje az összes $v \in G$ csúcs.

Az (u, v) pár résztartalmazásba felvételéhez ellenőrizni kell még azt, hogy minden címkére teljesül-e, hogy v-nek van-e legalább annyi megfelelő címkéjű szomszédja, mint u-nak, ez a szabály a vágási szabály ($cutting\ rules$) nevet viseli.

Példa: Adott az alábbi két címkézett gráf (q és G)

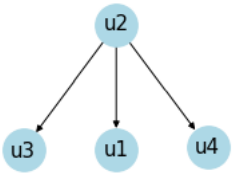


15. ábra: q gráf



16. ábra: G gráf

Először $V(q)$ csúcsok sorrendjét határozzuk meg. A gyökér meghatározásához megkeressük a legritkább címkét: G-ben minden címke (A, B és C) kétszer szerepel, a sorrend üres, azért a u_1, u_2, u_3 és u_4 marad a keresett csúcsok halmaza. A legnagyobb fokszámú ezek közül u_2, így az lesz a T fa gyökere. Szélességi keresést használunk (BFS), az új T fa nulladik szintje a gyökér, u_2, első szintje u_2 szomszédjai, ez esetben u_1, u_3 és u_4 is.



17. ábra: u_2 gyökerű BFS fa

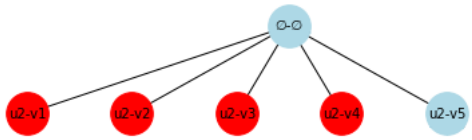
A T fa feldolgozását a nulladik szinttel kezdjük. Ekkor a legnagyobb konnektivitású, legnagyobb fokszámú, legkisebb címkéjű csúcspont az u_2, mert a nulladik szinten az az egyetlen csúcs. Így O sorrendbe bevesszük u_2-t. A nulladik szinten nincs több csúcs, léphetünk az első szintre. Itt a konnektivitás minden csúcsnál egyenlő, mindegyik szomszédos u_2-vel. A legnagyobb fokszám közülük u_1 és u_3, mindkét csúcs fokszáma 2. A két csúcs címkéje közül egyik sem a sorrendben lévő csúcs címkéje, így a gyökér meghatározásához hasonlóan az előfordulás száma 2-2 marad. Választanunk kell a két csúcs közül, legyen ez most u_1, bevesszük az O sorrendbe. Mivel nem fogytak el a szinten még a csúcsok, nem lépünk szintet. Újra kiválasztjuk a maradék csúcsok közül a legnagyobb konnektivitással rendelkezőt, ez most u_3, mert szomszédos u_1-gyel és u_2-vel is, míg u_4 csak u_2-vel szomszédos. Csak u_3 szerepel a halmazban, a legnagyobb fokszámú és legritkább címkéjű így u_3 marad. Bevesszük az O sorrendbe. Még van

sorrendben nem szereplő csúcs a szinten, így nem lépünk tovább. A maradék csúcsok közül (csak u_4) a legnagyobb konnektivitású u_4, a legnagyobb fokszámú és legritkább címkéjű is. u_4-et bevesszük az O sorrendbe. Minden $V(q)$ csúcs a sorrendben van, így végeztünk.

A következő sorrendet kaptuk: $O = \{u_2, u_1, u_3, u_4\}$

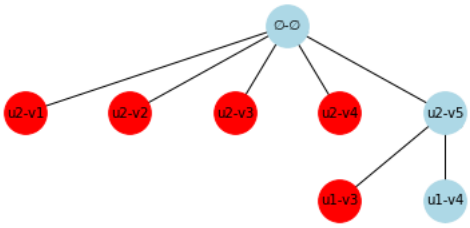
A csúcsokat ennek a sorrendnek megfelelően fogjuk tehát bevenni a résztartalmazásba.

A kezdő résztartalmazás üres, ezért a sorrendben első u_2 csúcshoz minden $v_j \in V(G)$ jelölt lesz. A vágási szabályokat nézve a következőket állapíthatjuk meg: u_2-nek három szomszédja van: kettő B címkéjű, egy A. v_1-nek egy szomszédja van, amely A címkével rendelkezik, ezért az (u_2, v_1) pár nem megfelelő. v_2-nek három szomszédja van, egy A, egy B és egy C címkéjű, B címkéjű szomszédból nincs elég, így az (u_2, v_2) pár sem megfelelő. v_3-nak két A és egy C címkéjű szomszédja van, az (u_2, v_3) pár sem jó a résztartalmazás bővítéséhez. v_4 szomszédja egy A, egy B és egy C címkéjű csúcs, B címkéjű szomszédból most sincs elég, így az (u_2, v_4) pár sem megfelelő. v_5-nek egy A és kettő B címkéjű szomszédja van, ez megegyezik u_2 csúcsainak címkéjével, illetve azok számával, ezért a (u_2, v_5) jó pár, bővítjük a résztartalmazást.



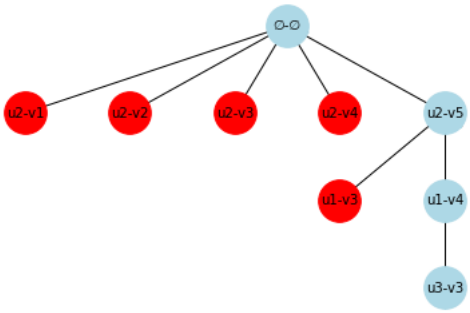
18. ábra: u_2 csúcshoz az első jó jelölt v_5

Következő csúcs a sorrendben u_1, amelynek egy résztartalmazásban szereplő szomszédja van, u_2. u_1 jelöltjei azon $v_j \in V(G)$ csúcsok lesznek, amelyek szomszédosak u_2 párjával, v_5-tel. Így tehát a jelöltek: v_3, v_4, v_6. A vágási szabályok következnek. u_1 szomszédja egy B és egy C címkéjű csúcs. v_3 nem lesz jó jelölt, mert két A és egy C címkéjű szomszédja van, a B címke hiányzik. v_4-nek viszont egy A, egy B és egy C szomszédja van, így megfelelő pár u_1-nek, felvesszük a résztartalmazásba (u_1, v_4)-t.



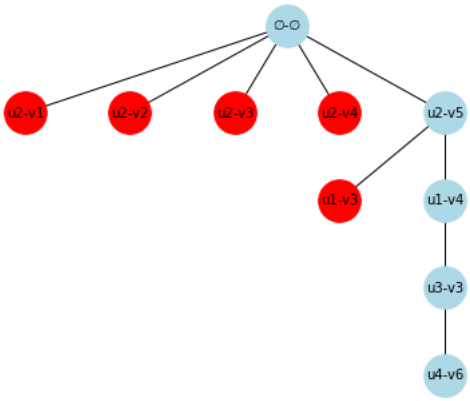
19. ábra: az (u_2, v_5) résztartalmazást az (u_1, v_4) párral bővíthetjük, (u_1, v_3) párral nem

A harmadik csúcs u_3, melynek résztartalmazásban szereplő szomszédjai u_1 és u_2. Ekkor a jelölteket úgy választjuk ki, hogy megnézzük az egyik résztartalmazásban szereplő szomszéd párjának szabad szomszédjait, majd ellenőrizzük, hogy az a többi résztartalmazásban szereplő szomszéd párjával is szomszédos-e. Nézzük tehát u_1 párjának, v_4-nek résztartalmazásban nem szereplő szomszédjait, ezek v_2 és v_3. Erre a két csúcsra kell ellenőriznünk, hogy u_2 párjával, v_5-tel szomszédosak-e. v_2 nem, v_3 igen. Így u_3-nak egyetlen jelöltje marad, v_3. Ellenőrizzük a vágási szabályokat: u_3 szomszédja egy A és egy C címkéjű csúcs, v_3 szomszédja pedig két A és egy C címkéjű csúcs. A szabály szerint felvehetjük a résztartalmazásba az új (u_3, v_3) párt.



20. ábra: A résztartalmazást (u_3, v_3) párral bővíthetjük

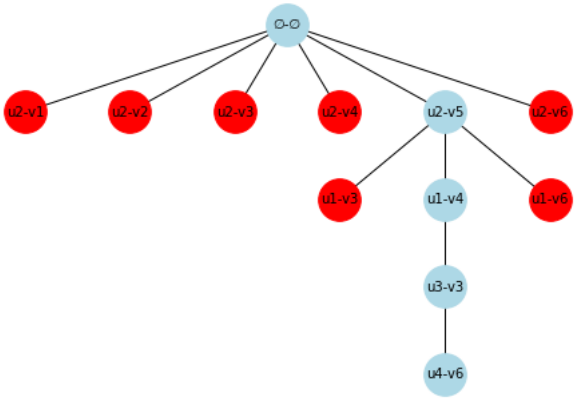
Utolsó csúcs a sorrend szerint u_4, jelöltjeinek meghatározásához meg kell néznünk résztartalmazásban szereplő szomszédjait. Egy szomszédja szerepel a résztartalmazásban, u_2, mely párjának, v_5-nek megkeressük szabad szomszédjait, amely csak a v_6 csúcs. Ellenőrizzük a vágási szabályt, u_4-nek egy C címkéjű szomszédja van, v_6-nak is, tehát felvehetjük a résztartalmazásba az (u_4, v_6) párt. Megtaláltunk egy tartalmazást.



21. ábra: Egyelten új jelöltpár van az ágon, ez az (u_4, v_6) pár

A fában visszalépve látjuk, hogy több tartalmazás nincs (u_1 kimaradt jelöltje, v_6 nem felel meg a vágási szabályoknak, v_6 ellenőrzése u_2 jelöltjeként is hasonló eredményt ad). Így az algoritmus leáll, megkaptuk, hogy q G egy részgráfjával izomorf, a következő párok lesznek: (u_1, v_4), (u_2, v_5), (u_3, v_3) és (u_4, v_6).

A keresési fában pirossal jelöltek azok a csúcsok, amelyek a vágási szabály miatt nem megfelelőek.



22. ábra: A VF2++ algoritmus során előállt keresőfa

3.5 DAF algoritmus

A DAF algoritmust Myoungji Han, Hyunjoon Kim, Geonmo Gu, Kunsoo Park és Wook-Shin Han publikálta 2019-ben [10]. Az algoritmus dinamikus programozást használ, amellyel sokat gyorsított a régebbi részgráf kereső algoritmusokon nagy gráfok esetén. Az algoritmust leíró cikkben irányítatlan, címkézett gráfok szerepelnek.

Az algoritmus három fő részből áll:

(1) DAG-Graph DP: irányított körmentes gráf segítségével $Candidate\ Space$ (CS) felépítése és finomítása dinamikus programozással

(2) Adaptive Matching Order with DAG Ordering: adaptív párba rendezés

(3) Pruning by Failing Sets: nem megfelelő halmazok levágása az algoritmus gyorsítása érdekében

Első lépésként a q gráfból egy irányított körmentes gráfot (DAG) kell készítenünk, ehhez egy gyökeret választanunk a következőképpen: elsőként minden $u \in V(q)$ csúcshoz meghatározzuk a kezdeti jelölteket:

$$C_{ini}(u) = \{v |\ v \in V(G) \wedge deg_G(v) \geq deg_q(u) \wedge L_G(v) = L_q(u)\}$$

Tehát egy u csúcs jelöltjei azok a $v \in V(G)$ csúcsok lesznek, amelyekre igaz, hogy fokszámuk legalább akkora, mint u csúcsé, valamint a két csúcs címkéje megegyezik.

Miután a kezdeti jelölteket meghatároztuk, megkeressük a DAG gyökerét. Azt a csúcsot választjuk, amelyre a következő hányados értéke a legkisebb (ha több ilyen van, akkor bármelyik választható):

$$r \leftarrow \arg\min_{u \in V(q)} \frac{|C_{ini}(u)|}{deg_q(u)}$$

A gyökér az az u csúcs lesz, ahol a kezdeti jelöltek számát elosztva az u fokszámával q-ban a legkisebb számot kapjuk.

Az így kiválasztott gyökértől kezdve szélességi bejárást (Breadth-first search, BFS) folytatunk q-n, minden élt irányítottá teszünk úgy, hogy a magasabb szintről az alacsonyabb szint az irány, azonos szinten pedig szabadon választható. Ez a gráf lesz q_D. Előállítjuk q_D^{-1}-et, úgy, hogy q_D^{-1} csúcsai megegyeznek q_D csúcsaival, és q_D^{-1} csúcsai között lévő élek ellentétes irányúak q_D csúcsai közötti élekkel.

Ezután a CS (Candidate Space) felépítése és egyszerűsítése következik, a keresést ezen fogjuk végezni. CS tartalmazza azon $C(u)$ jelölteket minden $u \in V(q)$-hoz, amelyek az alábbi két feltételt kielégítik:

(1) $u \in V(q)$-hoz tartozó jelöltek halmaza: $C(u)$ - candidate set -, részhalmaza $C_{ini}(u)$-nak, vagyis a kezdeti jelöltek halmazának

(2) Akkor és csak akkor van él $v \in C(u)$ és $v' \in C(u')$ között, ha $(u, u') \in E(q)$ és $(v, v') \in E(G)$

CS egyszerűsítésének célja, hogy $C(u)$-t a lehető legkisebb méretűre csökkentsük. Mivel később CS-ben fogunk keresni, ezért így gyorsíthatunk az algoritmuson.

Először $C(u) = C_{ini}(u)\ \forall\, u \in V(q)$-ra. q_D^{-1}-t és q_D-t felváltva fogjuk nézni, mindig az aktuális lesz a q'. CS finomításánál minden $C(u)$-ból $C'(u)$-t készítünk dinamikus programozás segítségével.

$v \in C'(u)$ akkor és csak akkor, ha $v \in C(u)$ és létezik v-vel szomszédos v_c csúcs úgy, hogy $v_c \in C(u_c)$ u minden u_c gyerekére q'-ben, tehát v jelölt u minden q'-beli gyerekének legalább egy jelöltjével össze van kötve.

Eszerint q' csúcsait visszafelé fogjuk bejárni, így minden csúcs azután fog következni, miután a gyerekei már sorra kerültek. A harmadik finomítás után már általában nem lehet tovább csökkenteni CS méretét.

CS csúcsainak egyszerűbb leírásához a következő jelölés használatos: $\forall\, v \in C(u)$-ra és $\forall\, (u, u_c) \in E(q_D)$-re: $N_{u_c}^{u}(v)$-be azon v_c csúcsok tartoznak, amelyek G-ben szomszédosak v-vel és $v_c \in C(u_c)$. Ez a jelölés egy szomszédossági listának felel meg.

Ezután CS-ben megkeressük q minden tartalmazását, ez megfelel annak, mintha G-ben keresnénk. Ehhez először a kiválasztott r gyökér csúcsot hozzárendeljük egyik $C(r)$ jelöltjéhez. Egy $u \in q_D$ akkor bővíthető (extendable) egy M résztartalmazás esetén, ha u minden szülőjéhez választottunk párt, azaz egy v csúcsot G gráfból. Először azokat a csúcsokat nézzük meg, amelyeknek nincsen gyerekei. Minden bővíthető csúcshoz meghatározhatjuk a bővíthető jelöltek listáját a következőképpen: tegyük fel, hogy M résztartalmazásunk van, u szülei q_D-ben $p_1, \dots, p_k$ ezeknek a csúcsoknak már van $v \in G$ párja, mert u csak ekkor lehet bővíthető. Az u csúcs bővíthető jelöltjeinek halmazát M-hez úgy definiáljuk, hogy:

$$C_M(u) = \cap_{i=1}^{k} N_u^{p_i}(M(p_i))$$

Tehát u csúcs minden szomszédjának párjával szomszédos jelölteket keressük.

Minden $v \in G$ csúcshoz nyilvántartjuk, hogy párja-e valamely $u' \in q$ csúcsnak, ha igen, akkor megjelöljük, így tudjuk, hogy mely csúcsok nem okoznak problémát M bővítésénél.

Előfordul, hogy több u bővíthető csúcs is van, ekkor kétféle módszer van az új csúcs kiválasztására:

Jelölt-méret sorrend (Candidate-size order): azt választjuk, ahol $|C_M(u)|$ a legkisebb

Út-méret sorrend (Path-size order): azt választjuk, ahol $w_M(u)$ a legkisebb, $w_M(u)$ egy becslés az út tartalmazásokra

Az így kiválasztott u csúccsal és jelöltjeivel bővítjuk M-et, majd a részfa bejárása után visszalépünk.

Az algoritmus gyorsítható, ha megtaláljuk azokat a részeket, amelyeket feleslegesen járnánk be többször, tehát nem vezetnének teljes tartalmazás megtalálásához. A fát mélységi keresés (Depth-first search, DFS) sorrendjében járjuk be, tehát egy pár felvétele után a hozzá tartozó részfát szükséges bejárni, majd visszalépünk. Lehetséges azonban, hogy az az (u, v) pár, amelyet hozzávettünk, a felette lévő részfában másik párok miatt nem működik, így hiába keresünk az u csúcsnak más v' párokat, az algoritmus nem találna tartalmazást, ezért a sok új párt felesleges ellenőrizni. Ebben segít a $sikertelen\ halmazok$ $(failing\ sets)$ meghatározása. Jele: F_M, ahol M a fában egy út, amely egy résztartalmazásnak felel meg. Ezeket a halmazokat a fában lentről felfelé számítjuk ki, tehát a leveleket először. A meghatározáshoz szükséges a következő jelölés bevezetése: $anc(u)$ azon $u' \in V(q)$ csúcsok halmaza, amelyek q^D-ben u csúcs felmenői, illetve u csúcs maga. A levelek háromféle csoportba tartozhatnak:

(1) Konfliktus osztály: ha az (u, v) levélben v egy olyan csúcs, amely a levélhez tartozó útban (tehát résztartalmazásban) már egy másik u' csúcs párja. Ekkor ezt a levelet egy felkiáltójellel jelöljük (u, v)!. Ekkor $F_M = anc(u) \cup anc(u')$.

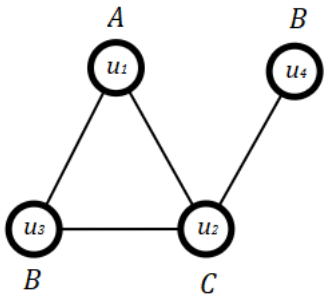
(2) Üres osztály: ha u csúcsnak nem jutott pár, ez a bővíthető jelöltek $(C_M(u))$ meghatározásánál fordulhat elő, ekkor a levelet $(u, \emptyset)$-ként jelöljük. Ekkor $F_M = anc(u)$.

(3) Tartalmazás osztály: ha a levéllel sikerült egy tartalmazást megtalálni. Ekkor $F_M = \emptyset$.

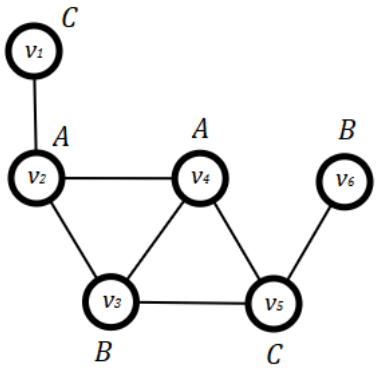
A többi párhoz, amelyek nem levelek, a levelek segítségével számoljuk ki F_M értékét. Ha a fa belső pontjának (most (u, v)) van olyan c gyermeke, amelynél $F_{M_c} = \emptyset$, akkor a ponthoz tartozó $F_M = \emptyset$. Ha nincs, akkor keresünk olyan c gyermeket, amelyhez tartozó F_{M_c}-ben nincs benne u, ha találunk ilyet, akkor $F_M = F_{M_c}$. Ha ilyen gyermek sem található, akkor $F_M = \bigcup_{c\ in\ children} F_{M_c}$.

Ha egy olyan (u, v) csúcshoz jutunk a fában, amelyhez tartozó F_M halmaz nem üres, és nem szerepel benne u, akkor az adott résztartalmazás nem u csúcs és párja miatt nem jó, tehát bármilyen új párt találnánk u-nak, ugyanúgy nem jutnánk tartalmazáshoz, tehát a csúcs minden testvére redundáns, így azokat nem kell bejárnunk. Ezzel megvalósul a fa vágása.

Példa: Adott az alábbi két gráf (q és G)



23. ábra: q gráf



24. ábra: G gráf

$$C_{ini}(u_1) = \{v_2, v_4\} \rightarrow \frac{|C_{ini}(u_1)|}{deg_q(u_1)} = \frac{2}{2} = 1$$

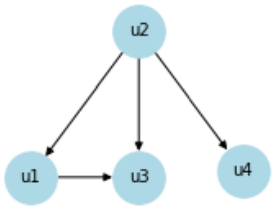
$$C_{ini}(u_2) = \{v_5\} \rightarrow \frac{|C_{ini}(u_2)|}{deg_q(u_2)} = \frac{1}{3} = 0,\dot{3}$$

$$C_{ini}(u_3) = \{v_3\} \rightarrow \frac{|C_{ini}(u_3)|}{deg_q(u_3)} = \frac{1}{2} = 0,5$$

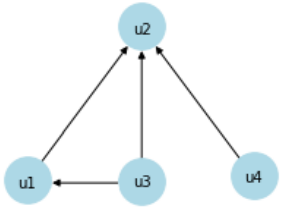
$$C_{ini}(u_4) = \{v_3, v_6\} \rightarrow \frac{|C_{ini}(u_4)|}{deg_q(u_4)} = \frac{2}{1} = 2$$

A legkisebb szám esetünkben a $0,\dot{3}$, így a gyökér u_2 lesz.

q-ból u_2-től kiindulva a következő gráfokat kapjuk: q_D és q_D^{-1}, amit q_D-ből éleinek megfordításával határozható meg.



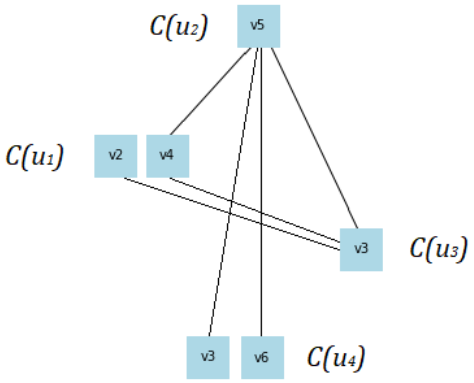
25. ábra: q_D gráf, amely q gráfból u_2-től indulva szélességi bejárással és a kimaradt élek felvételével kapható meg



26. ábra: q_D^{-1}, amelyet q_D-ből az élek megfordításával kapunk

CS csökkentéséhez q_D^{-1} és q_D gráfot kell használni felváltva. Kezdetben CS minden u csúcshoz a jelölteket tartalmazza.

CS eredetileg:



27. ábra: CS (Candidate Space) finomítások nélkül

Első finomítás: $q' = q_D^{-1}$

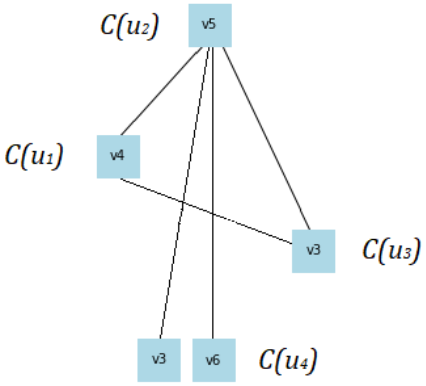
Első csúcs: u_2, mert nincs gyereke → $C'(u_2) = \{v_5\}$

Második csúcs: u_1, mert u_2 az egyetlen gyereke, és u_2 már sorra került, v_2 nincs összekötve u_2 jelöltjével, ezért kivesszük a jelöltek közül → $C'(u_1) = \{v_4\}$

Harmadik csúcs: u_4, mert gyerekei, u_2 már sorra került → $C'(u_4) = \{v_3, v_6\}$

Utolsó csúcs: $u_3 \rightarrow C'(u_3) = \{v_3\}$

CS az első finomítás után:

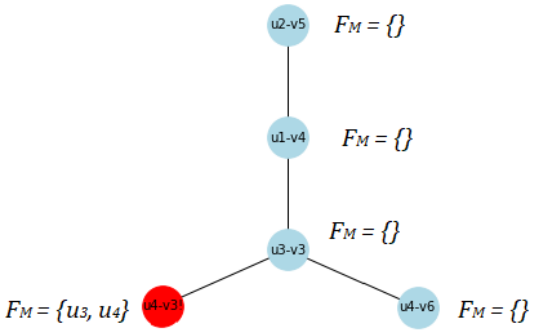


28. ábra: CS az első finomítás után

Ezután $q' = q_D$ következik, de már nem tudunk többet csökkenteni CS méretén.

A keresési fába először a gyökér, u_2 és jelöltjét vesszük fel, tehát az (u_2, v_5) párt. Ekkor a bővíthető csúcsok u_1 és u_4 lesznek, mert q_D-ben ennek a két csúcsnak egyetlen szülője u_2. Meghatározzuk a jó jelölteket: $C_M(u_1) = \{v_4\}$, $C_M(u_4) = \{v_3, v_6\}$, a jelölt-méret sorrend szerint u_1 csúccsal és párjával kell bővíteni a résztartalmazást: (u_1, v_4). Megnézzük, hogy van-e új bővíthető csúcs. Már u_3 minden szülője bekerült a résztartalmazásba, így jó csúcs. Jó jelöltjeinek halmaza: $C_M(u_3) = \{v_3\}$. A jelölt-méret sorrend szerint u_3 következik: felvesszük a fába (u_3, v_3) párt. Csak egy csúcs maradt hátra, u_4. $C_M(u_4) = \{v_3, v_6\}$, felvesszük az (u_4, v_3)! és az (u_4, v_6) párt a keresőfába. v_3 már szerepel a résztartalmazásban, ezért kell a felkiáltójel utána. (u_4, v_6) ágon azonban megtaláltunk egy részgráf tartalmazást. A sikertelen halmazok a résztartalmazás mentén üresek lesznek, (u_4, v_3)! levélnél pedig $F_M = \{u_3, u_4\}$. Ezen halmazok kiszámítása jelen esetben nem gyorsít az algoritmuson.

A keresés során előállt keresőfa F_M értékekkel:



29. ábra: A DAF algoritmus által előállított keresőfa

A DAF algoritmus segítségével megkaptuk, hogy q G egy részgráfjával izomorf, a csúcspárok a következők: (u_1, v_4), (u_2, v_5), (u_3, v_3) és (u_4, v_6).

4. Algoritmusok implementálása

A dolgozatban irányítatlan, címkézetlen gráfokkal fogok foglalkozni, a részgráf kereső algoritmusokat ezeken nézem meg. G gráf egy következő fejezetben bemutatott adathalmazokból választott nagy gráf, q pedig minden esetben egy háromszög (három csúcsú teljes gráf).

Az algoritmusokat Python nyelven valósítottam meg. A gráfokat a `NetworkX` könyvtár [11] `Graph()` függvényével hoztam létre. Csúcsokat a `Graph.add_node(node)`, éleket pedig a `Graph.add_edge(node, node)` függvénnyel lehet a gráfhoz adni. A háromszöget, tehát a q gráfot ezekkel a függvényekkel hoztam létre. G gráfhoz szükséges volt egy tsv (tabulátorral tagolt) fájlból beolvasás, ebben a fájlban egy sorban két csúcs szerepelt tabulátorral elválasztva, amely azt mutatta meg, hogy a két csúcs szomszédos. Ennek beolvasásához a `NetworkX read_adjlist(path)` függvényét használtam. A `NetworkX` könyvtárnak léteznek gráf generáló függvényei is, ezeket a saját gráfok előállításánál használtam.

q és G létrehozása

```
import networkx as nx

q=nx.Graph()
q.add_node(0)
q.add_node(1)
q.add_node(2)
q.add_edge(0,1)
q.add_edge(1,2)
q.add_edge(0,2)

fileG = open(fileName, 'rb')
G = nx.read_adjlist(fileG, nodetype=int)
```

A `NetworkX` könyvtárnak sok hasznos függvénye van: például az éleket a `Graph.nodes()`, a csúcsokat a `Graph.edges()`, egy v csúcs szomszédjait a `Graph.neighbors(v)` függvénnyel kapjuk meg.

4.1 Ullmann algoritmus

Az Ullmann algoritmus eredetileg címkézett gráfokra vonatkozik, így változtatni kellett rajta. Minden $u_i \in V(q)$ csúcsnál a $C(u_i)$ jelöltek meghatározásához csak a fokszámot kellett figyelembe venni. Az algoritmus először egy M^0 kompatibilitás mátrixot épít fel, melynek sorai q csúcsainak, oszlopai G csúcsainak felelnek meg. A

mátrix értékeit úgy töltjük fel, hogy ha $u_i \in V(q)$ csúcshoz $v_j \in V(G)$ csúcs jelölt, akkor $m_{i,j}^0 = 1$, ha v_j nem jelölt, akkor pedig $m_{i,j}^0 = 0$. A tartalmazás egy M mátrixban lesz, erre a mátrixra igaz, hogy minden sorban pontosan egy 1 érték szerepel, mivel minden u_i csúcshoz egy pár tartozik, és minden oszlopban maximum egy 1 érték szerepel, mert nem minden v_j csúcsnak lesz párja, ugyanis G egy részgráfját kerestük. A párokat úgy kapjuk meg, hogy ahol a mátrixban 1 szerepel, annak a sornak és oszlopnak megfelelő csúcsok lesznek egy (u_i, v_j) pár. M^0 felépítése után az algoritmus az O sorrendnek megfelelően megy végig a mátrix sorain. Jelen esetben háromszögeket kerestünk, melyben minden csúcs fokszáma 1, így a fokszám szerinti rendezést el lehetett hagyni, a sorrend a csúcsok sorrendje lett: u_0, u_1, u_2.

Ullmann algoritmus: M^0 felépítése: createM0()

```
for u in V(q):
    for v in V(G):
        if (deg(v) >= deg(u)): M0[u][v]=1
        else: M0[u][v]=0
```

A mátrixot rekurzió segítségével dolgoztam fel. Egy d mélységben előálló mátrix M^d. Ha d eléri q csúcsainak számát, akkor megtaláltunk egy tartalmazást. M^d mátrixban a d előtti sorokban egy 1 érték szerepel, utána lehetséges, hogy több, mivel a d utáni soroknak megfelelő csúcsok még nem kerültek sorra. M^d mátrixból M^{d+1} mátrixot úgy kapunk, hogy a d. sorban minden 1 értékre a következőket tesszük (legyen $m_{d,j}^d = 1$, tehát az oszlop száma j): d. sor minden $m_{d,k}^d$ $(j \neq k)$ elemét 0-ra vesszük, j. oszlop minden $m_{e,j}^d$ $(e \neq d)$ elemét szintén 0-ra vesszük. Ezzel megvalósítottuk a $szűrés$ részt, hiszen több csúcshoz nem lesz jelölt v_j. A $finomítás$ során ellenőrizni szükséges u_d szabad szomszédjainak jelöltjeit, ha valamely jelölt nem szomszédos v_j-vel, akkor a megfelelő helyen a mátrixban az 1-est 0-ra javítjuk.

Ullmann algoritmus: keresés: Search(M, d)

```
if (d == |V(q)|): tartalmazás
else:
     for v in M[d]:
          if (M[d][v] == 1):
               newM=copy(M)
               # filtering
               for otherV in V(G):
                    newM[d][otherV] = 0
               for otherU in V(q):
                    newM[otherU][v] = 0
               newM [d][v] = 1
               # refinement
               for n in neighbors(q, d):
```

```
                    for candidateV in V(G):
                         if (newM[n][candidateV] == 1):
                              if not (candidateV in neighbors(G,v)):
                                   newM[n][CandidateV] = 0

               Search(newM, d + 1)
```

4.2 VF2 algoritmus

A VF2 algoritmus leírásában irányított, címkézetlen gráfok szerepelnek, így az algoritmust át kellett alakítani, hogy irányított gráfokon használhassuk. Ez az átalakítás a jelöltek kiszámításán és a megvalósíthatósági szabályokon történt. Egy s résztartalmazáshoz azon szabad u és v csúcsok lesznek a jelöltek, amelyek szomszédosak valamely résztartalmazásban szereplő csúccsal, mivel a gráfban irányítatlan élek vannak, ezért nem különböztetjük meg a szülő-gyermek kapcsolatot. E módon meghatározott halmazokra a $T_q(s)$ és $T_G(s)$ jelöléseket használjuk. Ha közülük valamelyik üres, akkor minden szabad u és v csúcs jelölt lesz. A megvalósíthatósági szabályoknál s állapotban (u, v) új pár esetén ellenőrizni szükséges, hogy u minden résztartalmazásban szereplő szomszédjának párja szomszédos-e v-vel, és fordítva, v minden résztartalmazásban szereplő szomszédjának párja szomszédos-e u-val. Ez az R_{pred} és R_{succ} szabályoknak felel meg. R_{in} és R_{out} helyett ellenőrizzük, hogy v-nek van-e legalább annyi $T_G(s)$-ben szereplő szomszédja, mint u-nak $T_q(s)$-ben szereplő szomszédja. Utolsó szabály, R_{new} arra módosul, hogy u $\widetilde{N}_q(s)$-beli szomszédjainak és v $\widetilde{N}_G(s)$-beli szomszédjainak számát nézzük, ekkor $\widetilde{N}_q(s)$ és $\widetilde{N}_G(s)$ azon csúcsok halmaza, amelyek nem szerepelnek résztartalmazásban, és nem szomszédosak valamely résztartalmazásbeli csúccsal.

A résztartalmazást a programban M jelöli. Egy u csúcs párját a pszeudokódban `pair(u)` jelöli, v párját pedig `pair(v)`. Az u és v jelölteket két listában tároltam. Minden résztartalmazásban szereplő csúcs szabad szomszédjait megkeresem, ezeket a jelöltekhez adom. Ha valamelyik lista üres, akkor minden szabad csúcs jelölt lesz.

```
VF2 algoritmus: jelöltek kiszámítása: findCandidates(M)
uCandidates = []
vCandidates = []
for un in Mq:
     for uc in neighbors(q, un):
            if (uc not in Mq) & (uc not in uCandidates):
               uCandidates.append(uc)
for vn in MG:
     for vc in neighbors(G, vn):
            if (vc not in MG) & (vc not in vCandidates):
```

```
                vCandidates.append(vc)
if |uCandidates| == 0 or |vCandidates| == 0 :
     uCandidates = V(q) - Mq
     vCandidates = V(G) - MG
return uCandidates, vCandidates
```

Ezután u jelöltek közül ki kell választani a sorrend szerinti legelsőt, ez legyen a listában az első elem. A három fenti megvalósítási szabály a `firstRule, secondRule, thirdRule` nevet viseli. Ha az egyik feltétel nem teljesül, akkor nem szükséges megnézni a többit, ezt felhasználva sokat gyorsult az algoritmus.

```
A megvalósítási szabályok ellenőrzése: feasibility(M, newU, newV)
firstRule = True
secondRule = True
thirdRule = True
for uNeighbor in Mq ∩ neighbors(q, newU):
     if not (pair(uNeighbor) in neighbors(G, newV)):
          firstRule = False
for vNeighbor in MG ∩ neighbors(G, newV):
     if not (pair(vNeighbor) in neighbors(q, newV)):
          firstRule = False
if (FirstRule):
     cardq = | Tq ∩ neighbors(q, newU) |
     cardG = | TG ∩ neighbors(G, newV) |
     if cardG < cardq:
          secondRule=False
if (FirstRule & SecondRule):
     Nq = V(G) - MG - TG
     NG = V(q) - Mq - Tq
     cardq = | Nq ∩ neighbors(q, newU) |
     cardG = | NG ∩ neighbors(G, newV) |
     if cardG < cardq:
          thirdRule=False
return (FirstRule & SecondRule & ThirdRule)
```

Az algoritmus rekurzión alapul. A függvényben szükséges `Mq, MG, Tq` és `TG` karbantartása az aktuális állapotnak megfelelően.

```
Algoritmus: vf2(M)
if |M| == |V(q)|: tartalmazás
else:
     uCandidates, vCandidates = findCandidates(M)
     newU = uCandidates[0]
     for newV in vCandidates:
          feasibilty = feasibility(s, newU, newV)
          if feasibility:
               NewM=M.copy()
               NewM[u]=v
               vf2(NewM)
```

4.3 VF2++ algoritmus

A VF2++ algoritmust át kellett alakítani, ugyanis eredetileg irányítatlan, címkézett gráfokra szólt. Így a címkék ellenőrzését kivettem az algoritmusból. $V(q)$ csúcsok sorrendjét egy listában tároltam, amely felépítésének menete a következő volt: a gyökér meghatározásához a csúcsok fokszámaiból kiválasztottam a legnagyobbat. Ha több csúcs volt, amelynek fokszáma a maximum, akkor ezen csúcsok közül az első lett a fa gyökere. q gráfból a szélességi bejárást a `networkx.bfs_tree(q, root)` függvény segítségével valósítottam meg, ezután egy adott szinten található csúcsokat a `nx.descendants_at_distance(tree, root, depth)` függvénnyel kaptam meg.

A sorrend felépítése: matchingOrder(q):

```
order = []
while |order| ≠ |V(q)|:
     maxDegreeNodes = arg max deg (V(q) – O)
     root = maxDegreeNodes[0]
     tree = nx.bfs_tree(q, root)
     for depth in range(|V(q)|):
          VdSet = nx.descendants_at_distance(tree, root, depth)
          Vd = list(VdSet)
          while |Vd| > 0:
               maxConnectivityNodes = arg max neighbors in order (Vd)
               maxDegNodes = arg max deg (maxConnectivityNodes)
               newNodeInOrder = maxDegNodes[0]
               order.append(newNodeInOrder)
               Vd.remove(newNodeInOrder)
```

A sorrend felépítése után ennek megfelelően kell az aktuális csúcs jelöltjeit meghatározni, ehhez a csúcs egyik résztartalmazásban szereplő szomszédja párjának szomszédjaira megnézzük, hogy melyek szomszédosak a csúcs összes többi résztartalmazásbeli szomszédjának párjával. Ha a résztartalmazásban nincs a csúcsnak párja, akkor minden szabad G gráfbeli csúcs jelölt lesz. Ez az első lépésnél fordulhat elő összetett gráfok esetén (mivel a sorrendet szélességi bejárás szerint állítottuk fel, nincs olyan csúcs az elsőn kívül, amelynek ne lenne szomszédja a résztartalmazásban). A jelölteket egy listában tároltam (`vCandidates`). A résztartalmazást egy `M` listában tároltam, melynek legyen i indexű eleme v. Ez azt mutatja meg, hogy u_i-hez v csúcs tartozik. Sorban haladva a listában egy `depth` mélységben járva látható, hogy a `depth` előtti csúcsoknak van párja, a `depth` utániaknak nincs. Így egy `depth` helyen szereplő u csúcs résztartalmazásbeli szomszédja egy olyan szomszéd lesz, amelynek indexe a sorrendben kisebb, mint `depth`. Résztartalmazásbeli szomszédot úgy talál a függvény,

hogy u csúcs szomszédjait nézve ellenőrzi, hogy indexe kisebb-e. Ha igen, akkor jelölteket keres, a végén pedig `break`-kel megállítja a szomszédok további keresését.

```
Jelöltek keresése: findCandidates(d)
u = order[d]
vCandidates = []
if d == 0:
     vCandidates = V(G)
else:
     for uNeighbor in neighbors(q, u):
          if order.index(uNeighbor) < d:
               vPairOfUNeighbor = M[uNeighbor]
               for vc in neighbors(G, vPairOfUNeighbor):
                    goodCandidate = True
                    for idx in range(d):
                         if order[idx] in neighbors(q, u):
                              otherVPair = M[order[idx]]
                              if not otherVPair in neighbors(G, vc):
                                   goodCandidate=False
                              if M[order[idx]] == vc:
                                   goodCandidate = False
                              if goodCandidate:
                                   vCandidates.append(vc)
               break
return vCandidates
```

Az algoritmusban a vágási szabályok is módosultak: u és a jelölt v szomszédjainak számát kell csak ellenőrizni, ha nincs v-nek legalább annyi szomszédja, mint u-nak, akkor v nem jó jelölt. A részgráf keresést rekurzívan valósítottam meg.

```
VF2++ algoritmus a sorrend meghatározása után: vf2pp(d):
if d == |V(q)|: tartalmazás
else:
     u = order[d]
     vCandidates = findCandidates(d)
     for v in vCandidates:
          if |neighbors(q, u)| ≤ |neighbors(G, v)|:
               M[u] = v
               vf2pp(d+1)
```

4.4 DAF algoritmus

A DAF algoritmus címkézett gráfokra vonatkozik, így szükséges volt az átalakítás. A kezdeti jelöltek meghatározásánál a címkéket nem kell figyelembe venni. A programban a jelölteket egy `set`-ben tároltam.

```
Kezdeti jelöltek (C_ini(u)) meghatározása
Cini = {}
for u in V(q):
     candidates = []
     for v in G.nodes():
     if (deg(u) <= deg(v)):
          candidates.append(v)
```

```
     Cini[u] = candidates
```

Ezután a fa gyökerének kiválasztásához minden csúcsra meg kellett határozni a megfelelő hányadost.

```
Gyökér kiválasztása: findRoot()
r = 0
min = 0
for u in V(q)
     CandidatesToDegRatio = |Cini[u]| / deg(u)
     if (min == 0):
          min = CandidatesToDegRatio
          r = u
     if (min > CandidatesToDegRatio):
          min = CandidatesToDegRatio
          r = u
```

Az így kiválasztott gyökértől szélességi kereséssel kapható q_D. A szélességi keresés azonban nem hagy a gráfban minden csúcsot, így be kellett rajzolni azokat, amelyek kimaradtak. q_D-ből az élek megfordításával kapható q_D^{-1}, amelyet a kódban `qDR` jelöl. Élek megfordítására a `NetworkX DiGraph.reverse(Graph)` függvény használható.

```
q_D és q_D^-1 meghatározása
qD = nx.algorithms.traversal.bfs_tree(q, source = r)
for u in V(q):
        for n in neighbors(q,u):
             if not (qD.has_edge(u, n) | qD.has_edge(n, u)):
                  qD.add_edge(u, n)

qDR = nx.DiGraph.reverse(qD)
```

A CS felépítéséhez a minden u csúcs jelöltjeit kellett ellenőrizni q_D vagy q_D^{-1} fordított topologikus sorrendje szerint.

```
CS csökkentése: refine(qC)
order = list(reversed(list(nx.topological_sort(qC))))
for u in order:
        for v in Cini[u]:
             for uc in qC.neighbors(u):
                  goodV = False
                  for vc in Cini[uc]:
                       if G.has_edge(v,vc):
                            goodV = True
                  if not goodV:
                       Cini[u].remove(v)
```

CS csökkentése után a csúcsokat egy `set`-ben tároltam, amelyben egy q_D-ben lévő (u, u_c) élhez hozzárendeltem egy `set`-et, amely u jelöltjeihez azokat a csúcsokat adta meg u_c jelöltjei közül, amelyek szomszédosak.

Jelöltek tárolása:

```
adjacencySet = {}
for e in E(qD):
        u = e[0]
        uc = e[1]
        NUUc = {}
        for v in Cini[u]:
             edgesFromV = []
             for vc in Cini[uc]:
                  if G.has_edge(v,vc):
                       edgesFromV.append(vc)
             NUUc[v] = edgesFromV
        adjacencySet[e] = NUUc
```

A keresési fában lévő csúcsokhoz tartozó F_M értékek kiszámításához hasznos tárolni minden $u \in V(q)$ csúcs őseit, így nem kell mindig kiszámolni. Ehhez egy `set`-et használtam.

Szülők és a csúcs összegyűjtése: createAnc()

```
ancestors = {}
for u in V(q):
parents = []
parents.append(u)
for p in predecessors(qD, u):
        parents.append(p)
ancestors[u] = parents
```

A keresés egy rekurzív keresés, amelynél az első lépésben a q^D gyökerének minden jelöltjével bővítjük a fát, majd minden ágon meghatározzuk az új jelölteket, amelyekkel tovább bővítünk. Az algoritmushoz továbbá szükséges a már párban lévő $v \in V(G)$ csúcsok nyilvántartása, ezek a `visitedNodes` listában szerepelnek. Az `extendableVertices set` egy adott u csúcshoz tartozó bővíthető jelölteket adja meg.

Visszalépéses keresés: backtrack(M, extendableVertices)

```
if |M| == |V(q)|: tartalmazás
        # failing set: tartalmazás osztály
if |M| == 0:
        for v in Cini[r]:
             visitedNodes.append(v)
             M[r] = v
             for c in neighbors(qD, r):
                  parents = []
                  if |predecessors(qD, c)| == 1:
                       extendableVertices[c] = adjacencySet[r,c][v]
             extendableVertices = sort by length (extendableVertices)
             backtrack(M, extendableVertices)
             visitedNodes.remove(v)

else:
        u = list(extendableVertices.keys())[0]
        if |extendableVertices[0]| == 0:
             # failing set: üres osztály
        for v in extendableVertices[u]:
```

```
            if v in visitedNodes:
                 # failing set: konfliktus osztály
            if v not in visitedNodes:
                 visitedNodes.append(v)
                  M[u] = v
                 extendableVertices.pop(u)
                 for c in qD.neighbors(u):
                      parents = predecessors(qD,c)
                      if all parents in M.keys():
                           extendableCandidates = adjacencyList[u, c][v]
                           for p in qD.predecessors(c):
                                extendableCandidates = extendableCandidates
                                           ∩ adjacencyList[p, c][M[p]]
                           extendableVertices[c] = extendableCandidates
                 extendableVertices = sort by length(extendableVertices)

                 backtrack(M, extendableVertices)
                 visitedNodes.remove(v)
                 M.pop(u)
```

F_M kiszámításához a levelek esetén meghatározhatók az értékek: üres osztály esetén $F_M = anc(u)$, konfliktus osztálynál $F_M = anc(u) \cup anc(u')$, tartalmazás osztálynál pedig $F_M = \emptyset$. Ezek meghatározhatók az algoritmus során a keresésben kommentezett részeknél. Nem levél esetén F_M a csúcspont c gyermekeinek F_{M_c} értékeiből számítható ki. A függvény rögtön visszatér a halmazzal, ha megtalálta, így a lehető leggyorsabban lehet kideríteni.

Egy csúcshoz tartozó F_M kiszámítása: failingSet(nodePair)

```
u = nodePair[0]
FM = []
for c in successors(qD, u):
        if |FMc| = 0:
             FM = []
             return FM
for c in successors(qD, u):
        if u not in FMc:
             FM = FMc
             return FM
FM = ⋃_{c in children} FMc
        return FM
```

5. Algoritmusok összehasonlítása

Ebben a fejezetben a bemutatott algoritmusok összehasonlítását végzem el nagy gráfokon, megvizsgálom az elméleti és gyakorlati különbségeket közöttük.

5.1 Adathalmaz bemutatása

Összehasonlítás során egy nagy gráfban háromszögeket kerestem. G gráf egy nagy gráf, a később bemutatott adathalmazokból, q pedig egy háromszög. A cél tehát egy gráfban háromszögek keresése, olyan három csúcs találása, amelyek egymással mind szomszédosak. Mivel a gráfok címkézetlenek, az algoritmusok egy háromszöget hatféleképpen találnak meg a három csúcs permutációja miatt ($3! = 6$), ezt figyelembe kell venni a talált háromszögek számának megadásánál. A háromszög az egyik legegyszerűbb részgráf, mégis sok területen hasznos: szociális hálóknál például egy ismerős két ismerőse gyakran ismerős lesz. A háromszög keresés használható még spam detektálásnál, tudományos fórumokban egy kérdés megválaszolásának esélyének becslésére, biológiában motívumok keresésére és folyamatok előrejelzésére. [12]

Az algoritmusokat többféle adathalmazon futtattam. Az első adathalmaz a Massachusetts Institute of Technology (MIT) és az Amazon Web Services által szervezett Graph Challenge verseny [13] során rendelkezésre bocsátott adathalmaz [14]. Ebben sokféle nagy gráf található. Az adathalmazok nevei úgy épülnek fel, hogy először az adat forrása, majd, ha szükséges, akkor a dátum vagy a gráf csúcspontjainak száma szerepel. A következő halmazokon teszteltem:

as: autonóm rendszerek hálózata

ca-HepTh: nagyenergiájú fizika elméleti témakörében kutatók kollaborációja az arxiv.org oldalon

ca-GrQc: általános relativitáselmélet témakörben kutatók kollaborációja az arxiv.org oldalon

oregon1, 2: autonóm rendszerek peering információja a University of Oregon Route Views projektjéből

p2p-Gnutella: Gnutella peer-to-peer hálózata

simulated_blockmodel_graph: kisebb méretű statikus gráf

static_lowOverlap_lowBlockSizeVar: alacsony átfedés, alacsony mértékű méret különbségek a gráfban

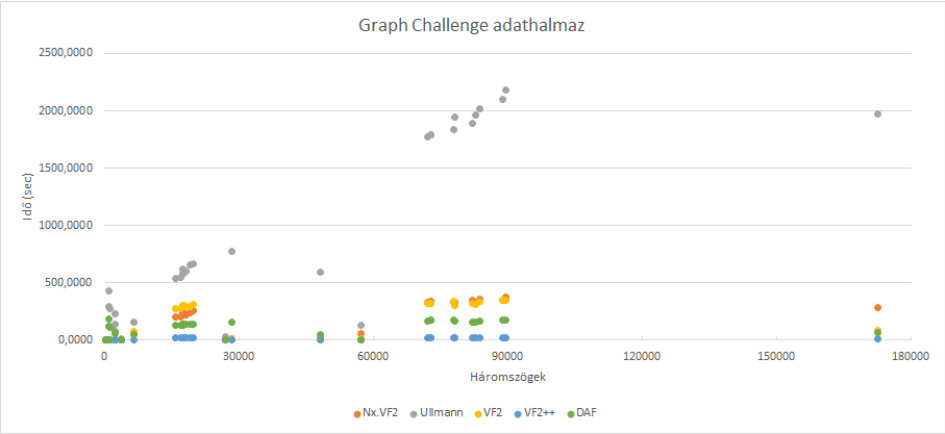
A fenti adathalmaz tulajdonságai a 1. táblázatban láthatók, az algoritmusok futási idejei pedig a 2. táblázatban.

Az algoritmusokat több különféle általam generált gráfokon is teszteltem, ez összehasonlításra különösen jó, ugyanis a gráfok alapfelépítése szinte azonos, így az algoritmusok tulajdonságai jobban mérhetők. Az elsőféle gráfhalmaz előállításához a `networkx gnm_random_graph(nodes, edges)` függvényét használtam. A csúcspontok számát mindegyik gráfhoz 1000-re választottam, az élek száma pedig 1500 és 50000 között 500-as léptékkel nőtt. Így 97 gráfot kaptam. A kapott gráfok csúcsait, éleit és a benne található háromszögek számát a 3. táblázat tartalmazza. A második gráfhalmazt a `networkx barabasi_albert_graph(nodes, m)` függvényt használtam, amely a Barabási-Albert modellnek [15] megfelelően hoz létre gráfokat, m azt mutatja meg, hogy a gráf felépítésekor egy új csúcs hány éllel csatlakozik régi csúcsokhoz. A gráfhalmazban a csúcsok száma 500 és 5000 közötti érték, amely 100-asával nőtt, m mindegyik gráfnál 3. A kapott 45 gráf csúcsait, éleit és a háromszögek számát a 4. táblázat tartalmazza.

5.2 Eredmények

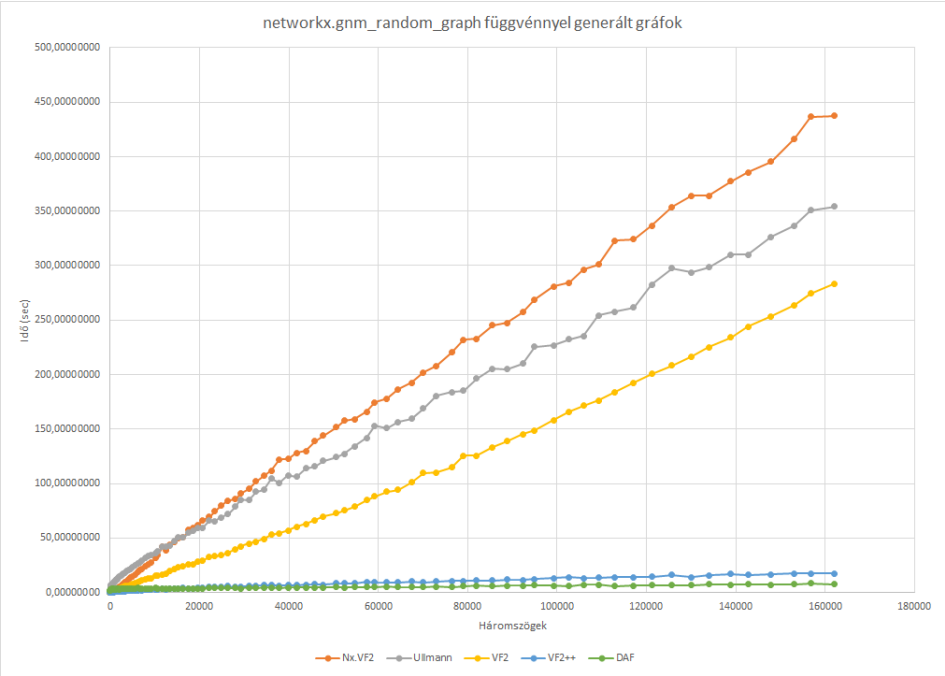
Az adathalmazokon a következő algoritmusok futási idejét vizsgáltam: NetworkX VF2, továbbá az általam programozott Ullmann, VF2, VF2++ és DAF algoritmust. A futási időket a `time.perf_counter( )` függvénnyel mértem.

A Graph Challenge adathalmazon a legjobban teljesítő algoritmus a VF2++ volt.



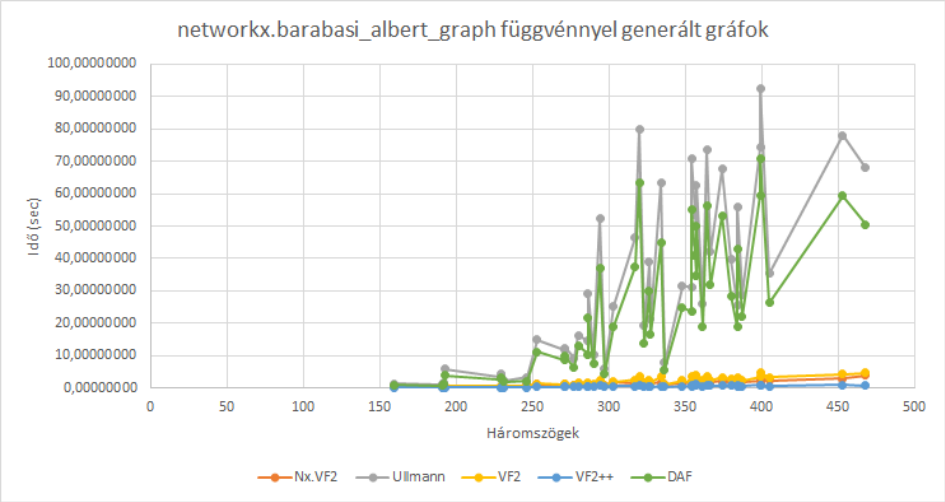
30. ábra: Algoritmusok eredménye a Graph Challenge adathalmazon

A `NetworkX gnm_random_graph` függvényével generált gráfok halmazán a DAF algoritmus mutatkozott a leggyorsabbnak, azonban a VF2++ hasonlóan gyors volt.



31. ábra: Algoritmusok eredménye a NetworkX gnm_random_graph függvényével generált gráfokon

A `NetworkX barabasi_albert_graph` függvényével generált gráfok esetében a VF2++ algoritmus volt a leggyorsabb.



32. ábra: Algoritmusok eredménye a NetworkX barabasi_albert_graph függvényével generált gráfokon

5.3 Összehasonlítás

Az algoritmusok időbeli komplexitása a következő:

Ullmann algoritmus esetén a legjobb $O(N^3)$, a legrosszabb pedig $O(N!\,N^2)$.

VF2 algoritmusnál legjobb esetben $O(N^2)$, legrosszabb $O(N!\,N)$.

VF2++ algoritmusnál a sorrend meghatározása $O(|V(q)|)$, egy u csúcshoz a jelöltek meghatározása: $O(deg(u))$, a vágási szabályok pedig u csúcsnál szintén $O(deg(u))$ komplexitásúak.

DAF algoritmuskor $DAG\ Graph\ DP$ résznél, tehát a CS felépítésénél és finomításánál maximum $O(|E(q)| \times |E(G)|)$, a keresés pedig maximum $\prod_{u \in V(q)} |C(u)|$, ami $O(|V(G)|^{|V(q)|})$.

Elméletben tehát Ullmann algoritmus lassabb, mint a VF2 algoritmus. VF2++ algoritmus gyorsabb, mint a VF2 algoritmus, ugyanis csak olyan új elemek szerepelnek az benne, amelyek gyorsabbak. A DAF algoritmus legkritikusabb része a CS felépítése, ezután a keresés gyors.

A különböző adathalmazokon futtatva az algoritmusokat a következő eredményeket figyeltem meg: Ullmann algoritmusa nagyobb gráfoknál mindig lassabb lesz, ezért inkább egyszerűbb gráfoknál, vagy olyan esetben hasznos, amikor nagyobb részgráfot keresünk, így egy csúcsnak nincs olyan sok jelöltje. Mivel háromszögeket kerestem a különböző gráfokban, amely egy egyszerű részgráf, ezt is figyelembe kellett venni az értékelés során. Az általam készített VF2 algoritmus legtöbbször gyorsabb volt, vagy közel azonos idő alatt futott le, mint a `NetworkX` könyvtárban található megvalósítás. Háromszögek vizsgálatakor a VF2 algoritmus nem teljesített annyira jól azoknál a gráfoknál, ahol sok közel azonos fokszámú csúcs található (`NetworkX gnm_random_graph` függvényével generált gráfok). VF2++ algoritmus szinte minden esetben a legjobban teljesített, észrevehető, hogy sokkal gyorsabb lett elődjénél, a VF2 algoritmusnál. A DAF algoritmus futási ideje nagyban függött vizsgált gráf tulajdonságaitól. A futási idők felbontásánál látható volt, hogy a legtöbb időt CS felépítése és csökkentése veszi el, a keresés ezután nagyon gyors. Ezért akkor ajánlott használni ezt az algoritmust, ha nagymértékben csökkenthető a CS. Ha csak néhány csúcsot tudunk eltávolítani CS-ből, akkor az az idő, amíg az algoritmus végignézi az összes csúcsot, több lesz, mint ha minden kezdeti jelöltet benne hagyott volna.

6. Összefoglaló és továbbfejlesztési lehetőségek

Dolgozatomban a részgráf keresést ismertettem. Bemutattam a részgráf-izomorfia problémakörét és az aktuális algoritmusokat, azokat több adathalmazon teszteltem. Részletes összefoglaló az algoritmusokról még nem volt fellelhető eddig. Különböző gráfok alapján arra a következtetésre jutottam, hogy Ullmann algoritmusának teljesítménye sok csúcspont esetén nagyon lecsökken, így többmilliós csúcspontú gráfoknál részgráf keresésre nem alkalmas. A VF2 algoritmus gráf felépítésétől függően különböző teljesítményű: nagy fokszámú csúcsok esetén lecsökken. A VF2++ algoritmus minden tesztelt adathalmazon a legjobb eredmények egyikét érte el. A DAF algoritmus olyan nagy gráfoknál javasolt, ahol a CS (Candidate Space) mérete nagymértékben lecsökkenthető, mert ennek felépítése és finomítása telik a legtöbb időbe, így nem nagy csökkenés esetén nem térül meg az erre fordított idő.

A részgráf-izomorfia sok területen, így például a bioinformatikában vagy a kémiában használható, segítségével különböző tulajdonságok, reakciók becsülhetők meg.

Továbbfejlesztési lehetőség az algoritmusok másféle részgráfokra történő tesztelése, illetve még nagyobb, többmillió csúcspontú gráfokra futtatása. Új módszerként ígéretes lehet mesterséges intelligencia módszereivel történő részgráf keresés is [16].

Irodalomjegyzék

[1] A. Recski, G. Y. Katona és C. Szabó, A számítástudomány alapjai, Typotex, 2002.

[2] G. Ivanyos, L. Rónyai és R. Szabó, Algoritmusok, Typotex, 1998.

[3] L. A. Hemaspaandra, "SIGACT News Complexity Theory Column 100".

[4] J. R. Ullmann, "An Algorithm for Subgraph Isomorphism," *Journal of the Association for Computing Machinery*, vol. 23, no. 1, pp. 31-42, 1976.

[5] J. R. Ullmann, "Bit-Vector Algorithms for Binary Constraint Satisfaction," *ACM Journal of Experimental Algorithmics*, vol. 15, no. 1, 2011.

[6] U. Cibej and J. Mihelič, "Improvements to Ullmann's Algorithm for the Subgraph Isomorphism Problem," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 29, no. 7, 2015.

[7] L. P. Cordella, P. Foggia, C. Sansone and M. Vento, "A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs," *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, vol. 26, no. 10, 2004.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, Introduction to Algorithms, MIT Press, 1990.

[9] A. Jüttner and P. Madarasi, "VF2++ - An Improved Subgraph Isomorphism Algorithm," Egerváry Research Group on Combinatorial Optimization, Budapest, 2018.

[10] M. Han, H. Kim, G. Gu, K. Park and W.-S. Han, "Efficient Subgraph Matching: Harmonizing Dynamic Programming, Adaptive Matching Order, and Failing Set Together," *SIGMOD*, 2019.

[11] "NetworkX - Network Analysis in Python," [Online]. Available: https://networkx.org/.

[12] M. N. Kolountzakis, G. L. Miller, R. Peng and C. E. Tsourakakis, "Efficient Triangle Counting in Large Graphs via Degree-based Vertex Partitioning".

[13] S. Samsi, V. Gadepally, A. Reuther and J. Kepner, "Subgraph Isomorphism Graph Challenge," Lincoln Laboratory - MIT, 2017.

[14] "Graph Challenge Data Sets," MIT, Amazon Web Services, [Online]. Available: http://graphchallenge.mit.edu/data-sets.

[15] A.-L. Barabási és R. Albert, „Emergence of Scaling in Random Networks," Science, 1999.

[16] Y. Zhang, Z. Yang and Z. Lou, "Subgraph Pattern Matching on Graphs with Deep Representations".

Függelék

1. táblázat: A Graph Challenge adathalmaz

Szám	Adathalmaz	Csúcsok	Élek	Háromszögek
1	simulated_blockmodel_graph_50_nodes	50	283	365
2	p2p-Gnutella04	10876	39994	934
3	simulated_blockmodel_graph_100_nodes	100	678	1006
4	p2p-Gnutella05	8846	31839	1112
5	p2p-Gnutella06	8717	31525	1142
6	p2p-Gnutella09	8114	26013	2354
7	p2p-Gnutella08	6301	20777	2383
8	static_lowOverlap_lowBlockSizeVar_1000_nodes	1000	7852	3888
9	as20000102	6474	12572	6584
10	oregon1_010407	10729	21999	15834
11	oregon1_010331	10670	22002	17144
12	oregon1_010505	10943	22607	17597
13	oregon1_010512	11011	22677	17598
14	oregon1_010428	10886	22493	17645
15	oregon1_010519	11051	22724	17677
16	oregon1_010414	10790	22469	18237
17	oregon1_010421	10859	22747	19108
18	oregon1_010526	11174	23409	19894
19	simulated_blockmodel_graph_500_nodes	500	8495	27096
20	ca-HepTh	9875	25973	28339
21	ca-GrQc	5241	14484	48260
22	simulated_blockmodel_graph_1000_nodes	1000	18773	57318
23	oregon2_010505	11157	30943	72182
24	oregon2_010512	11260	31303	72866
25	oregon2_010512	11260	31303	72866
26	oregon2_010428	11113	31434	78000
27	oregon2_010407	10981	30855	78138
28	oregon2_010421	11080	31538	82129
29	oregon2_010331	10900	31180	82856
30	oregon2_010519	11375	32287	83709
31	oregon2_010414	11019	31761	88905
32	oregon2_010526	11461	32730	89541
33	simulated_blockmodel_graph_5000_nodes	5000	99294	172714

2. táblázat: Graph Challenge adathalmaz eredményei

Szám	Nx.VF2	Ullmann	VF2	VF2++	DAF
1	0,1519	0,0510	0,0307	0,0117	0,0168
2	2,6173	426,8141	6,1503	2,0258	185,8218
3	0,4898	0,2485	0,0939	0,0331	0,0522

4	4,1739	295,2842	5,7167	1,5170	120,5087
5	3,6300	277,0070	4,9264	1,5361	113,9086
6	7,3236	233,5446	5,0286	1,3567	80,1095
7	6,7670	144,0292	4,2360	1,2681	54,2548
8	4,4093	13,8105	1,5565	0,5786	3,1644
9	65,6385	157,6907	77,0997	7,2696	50,3156
10	206,4691	540,1593	274,4702	19,0004	127,1932
11	205,1692	549,4367	272,3818	18,4431	129,5600
12	235,1611	623,6577	299,6730	19,3164	133,9383
13	231,4629	580,2591	302,5089	21,5304	137,7708
14	242,5116	607,8141	298,0588	19,4105	142,3534
15	231,3397	585,9478	306,9826	20,3039	141,8676
16	221,8715	599,0361	291,9871	18,8998	138,1939
17	242,2522	653,1530	297,3195	18,7377	135,4893
18	256,4757	662,6084	314,8125	21,9723	142,2548
19	24,9289	29,3723	4,0684	1,0484	1,5256
20	16,8850	776,8093	12,8252	1,3176	160,0808
21	23,5598	591,9085	6,5804	1,1426	49,4759
22	59,3795	128,8415	13,3039	2,8548	3,6684
23	332,2924	1773,6370	317,9350	24,7283	171,2371
24	335,6510	1789,9335	324,2827	21,4636	172,9627
25	335,6510	1789,9335	324,2827	21,4636	172,9627
26	333,8239	1835,0102	342,4767	21,1326	175,6130
27	329,2598	1940,8679	303,4775	23,4329	166,3570
28	346,8833	1892,0504	317,8198	20,4460	157,5543
29	325,8979	1961,3802	307,7355	21,9519	162,3219
30	354,2774	2018,6306	336,4435	23,2569	170,7948
31	350,7060	2100,3847	349,0344	25,2988	172,8470
32	375,6743	2184,8368	345,3560	23,7371	177,7041
33	286,0537	1970,0975	85,5107	17,2905	68,5058

3. táblázat: NetworkX gnm_random_graph függvénnyel generált gráfok adatai és eredmények

Szám	Csúcsok	Élek	Háromsz.	Nx.VF2	Ullmann	VF2	VF2++	DAF
1	1000	1500	9	0,0456	1,5780	0,0914	0,0275	1,7706
2	1000	2000	10	0,0545	2,1258	0,1018	0,0341	1,8816
3	1000	2500	16	0,0741	2,5410	0,1592	0,0460	2,2011
4	1000	3000	41	0,1061	2,9875	0,1830	0,0640	2,5338
5	1000	3500	57	0,1528	3,6416	0,2625	0,0859	2,6001
6	1000	4000	84	0,1895	4,2213	0,3121	0,1210	2,6868
7	1000	4500	99	0,2187	4,7797	0,5122	0,1556	2,8061
8	1000	5000	171	0,3912	6,4348	0,5557	0,2015	2,9133
9	1000	5500	216	0,4518	6,4631	0,7969	0,2518	2,9386
10	1000	6000	302	0,6104	6,4461	0,8657	0,3334	2,9781
11	1000	6500	367	0,8131	6,6476	0,9505	0,3029	3,2038
12	1000	7000	471	1,1433	7,6787	1,2706	0,3585	3,0962
13	1000	7500	583	1,2680	8,1631	1,4308	0,4371	2,9817

14	1000	8000	695	1,7976	8,8429	1,5893	0,4436	2,9523
15	1000	8500	811	1,9323	9,0997	1,8016	0,5352	3,0731
16	1000	9000	978	2,2678	9,6957	2,1519	0,5231	2,7312
17	1000	9500	1151	2,9782	11,6040	2,4952	0,6374	2,6396
18	1000	10000	1356	3,6790	11,6606	2,7755	0,7600	3,1068
19	1000	10500	1533	3,8545	11,9742	2,9843	0,8454	3,1166
20	1000	11000	1744	4,5854	13,5708	3,3372	0,8306	3,4341
21	1000	11500	2004	5,2555	14,5320	3,6859	1,0058	2,9585
22	1000	12000	2341	5,9893	15,5933	3,9748	1,1289	3,1309
23	1000	12500	2549	7,2323	16,5467	4,6812	1,0449	3,1137
24	1000	13000	2910	7,8101	17,3363	5,1940	1,3090	3,0483
25	1000	13500	3318	9,5867	18,0648	5,6546	1,2207	3,0319
26	1000	14000	3658	10,0629	19,9643	6,3838	1,4024	3,1281
27	1000	14500	4116	12,3243	21,0267	6,3568	1,4847	3,1807
28	1000	15000	4529	13,8611	21,0656	7,2782	1,4999	3,4423
29	1000	15500	4861	15,0363	23,1606	7,6919	1,5908	3,2181
30	1000	16000	5533	16,6371	24,6360	9,0125	1,8005	3,2263
31	1000	16500	6103	18,6559	26,8566	9,8542	2,0300	3,9350
32	1000	17000	6633	20,4721	27,4874	10,4038	2,2049	3,2150
33	1000	17500	7052	21,8481	28,9208	11,2607	2,0125	3,3858
34	1000	18000	7769	23,9475	31,3524	12,4029	2,2223	3,2534
35	1000	18500	8449	25,8340	33,3300	12,7897	2,3071	3,2118
36	1000	19000	9051	27,2358	34,3887	13,0639	2,6051	3,2533
37	1000	19500	10248	32,1611	36,0502	15,2512	2,6997	3,8852
38	1000	20000	10664	34,6301	37,4683	15,4583	2,5903	3,3033
39	1000	20500	11626	41,8113	42,0149	16,6728	3,0925	3,6036
40	1000	21000	12486	38,4732	41,9507	17,3677	2,9484	3,4466
41	1000	21500	13327	43,7819	42,8467	19,8340	3,3530	3,4927
42	1000	22000	14321	46,4886	47,5155	21,4251	3,7641	3,6096
43	1000	22500	15277	49,6448	50,3829	23,0456	3,7807	3,7310
44	1000	23000	16295	50,4339	50,2612	23,9799	4,0613	3,8689
45	1000	23500	17493	57,3534	54,7057	25,6804	3,5114	3,5107
46	1000	24000	18577	58,9391	56,9711	25,7738	3,8679	3,5793
47	1000	24500	19530	61,4496	59,6012	28,6710	4,0411	3,8584
48	1000	25000	20635	66,4132	59,3691	29,3126	4,1782	3,5181
49	1000	25500	22091	69,4701	66,0948	32,2614	4,7695	4,0560
50	1000	26000	23394	74,2837	65,5977	33,1077	5,1087	4,1061
51	1000	26500	24711	79,5764	68,8099	34,1596	4,9636	4,0574
52	1000	27000	26347	84,2847	72,0650	36,2004	5,6384	4,1267
53	1000	27500	28012	85,7134	78,8909	39,5363	4,9831	3,9940
54	1000	28000	29147	90,9918	85,1843	42,4698	5,5481	3,7204
55	1000	28500	31012	95,0024	84,8130	44,7127	5,8796	4,5543
56	1000	29000	32599	102,0566	92,4958	46,7451	6,2147	4,2384
57	1000	29500	34377	107,0506	94,2238	49,0367	6,8811	4,6735
58	1000	30000	36037	111,7110	104,4242	52,9258	6,5375	4,6636
59	1000	30500	37810	122,2377	100,5567	54,3708	6,2432	4,1227

60	1000	31000	39779	122,7293	107,4878	56,7535	6,8796	4,3713
61	1000	31500	41652	128,1592	106,4031	60,2878	6,6982	4,4711
62	1000	32000	43837	129,6485	113,8979	62,8016	7,0750	4,3608
63	1000	32500	45770	138,8013	115,6711	66,3214	7,3169	4,7169
64	1000	33000	47636	143,8435	120,9145	69,9533	7,1439	4,7306
65	1000	33500	50539	151,5335	124,1245	72,8890	8,3707	4,8315
66	1000	34000	52318	157,7757	127,1083	75,5408	8,2802	4,5406
67	1000	34500	54698	158,8189	134,2667	78,6877	8,3435	4,9189
68	1000	35000	57399	165,8822	141,7886	84,7751	9,4756	4,9249
69	1000	35500	59142	174,6265	152,8265	87,9909	9,3404	5,0296
70	1000	36000	61849	178,0194	150,9121	92,3650	9,3394	5,4666
71	1000	36500	64407	186,4619	156,6043	94,4022	9,2394	5,0315
72	1000	37000	67445	192,4064	159,6340	101,1732	9,9994	5,1348
73	1000	37500	70019	201,8992	168,9562	109,7575	9,2487	4,8677
74	1000	38000	73020	207,8593	180,6645	110,1292	9,9666	5,5359
75	1000	38500	76409	220,5155	183,9594	114,7397	10,6919	5,1449
76	1000	39000	79078	231,9157	185,3568	125,5232	10,5253	5,8577
77	1000	39500	81965	233,0199	196,3273	125,5961	10,8592	6,0281
78	1000	40000	85524	245,2592	205,5501	133,3022	11,1732	5,6119
79	1000	40500	88767	247,4479	204,9286	138,8195	11,8229	6,1923
80	1000	41000	92277	257,3451	210,1732	145,3145	11,2784	6,4123
81	1000	41500	94963	268,8383	225,4350	148,6594	12,2742	6,6139
82	1000	42000	99182	280,8312	227,0088	158,2027	13,0377	6,0345
83	1000	42500	102689	284,1057	232,2010	166,0579	13,9082	5,9538
84	1000	43000	105903	296,3399	235,5108	171,4768	12,9929	6,8880
85	1000	43500	109255	301,0181	254,0642	176,1925	13,7410	7,0386
86	1000	44000	112978	322,8108	257,6143	183,8330	13,8188	5,8848
87	1000	44500	117120	324,1036	261,7166	192,2959	13,9965	6,3000
88	1000	45000	121165	336,7673	282,5745	200,7063	14,4327	6,8581
89	1000	45500	125719	353,7869	297,4999	208,5039	16,1793	6,7156
90	1000	46000	130065	364,0052	293,7456	216,6584	14,0991	6,7072
91	1000	46500	133957	364,3240	298,6896	225,0127	15,6969	7,4066
92	1000	47000	138867	377,6259	310,1546	234,2755	16,9567	7,1019
93	1000	47500	142695	385,5825	310,0166	243,9769	16,1088	7,4027
94	1000	48000	147837	395,6405	326,3707	253,2256	16,6105	7,2372
95	1000	48500	153102	416,3235	336,7671	263,5204	17,3820	7,4811
96	1000	49000	156720	436,6842	351,0056	274,6182	17,3414	8,2218
97	1000	49500	162119	437,4265	354,2759	283,5065	17,6115	7,5650

4. táblázat: NetworkX barabasi_albert_graph függvénnyel generált gráfok adatai és eredmények

Szám	Csúcsok	Élek	Háromsz.	Nx.VF2	Ullmann	VF2	VF2++	DAF
1	600	1791	159	0,2787	1,2430	0,2216	0,0623	0,8662
2	500	1491	191	0,3296	1,0178	0,2132	0,0575	0,6245
3	700	2091	192	0,4019	1,6887	0,3384	0,0860	1,1735
4	1200	3591	193	0,5349	5,8008	0,5467	0,1672	3,9336
5	1000	2991	229	0,6027	3,4561	0,5494	0,1652	2,6199

6	1100	3291	229	0,5941	4,2500	0,5359	0,1468	3,2774
7	800	2391	231	0,4925	2,3384	0,4251	0,1297	1,8186
8	900	2691	246	0,6329	3,2513	0,5060	0,1175	2,1714
9	2000	5991	253	1,0506	14,9513	1,3097	0,2763	11,1840
10	1700	5091	271	0,9875	11,6285	1,0807	0,2840	8,5104
11	1800	5391	271	1,0776	12,2229	1,2113	0,2622	9,6631
12	1500	4491	277	0,9647	9,1868	0,9951	0,2544	6,2896
13	2100	6291	280	1,1647	16,3298	1,5832	0,3269	13,0490
14	1900	5691	286	1,1403	14,5683	1,2525	0,2814	10,2934
15	2800	8391	286	1,1828	29,1533	1,7273	0,4997	21,6385
16	1600	4791	290	0,9449	10,2597	1,1739	0,2998	7,3576
17	3700	11091	294	1,5517	52,5263	2,3129	0,6370	36,8860
18	1300	3891	297	0,9240	6,0610	0,7809	0,2205	4,5283
19	2400	7191	303	1,6954	25,3061	1,9770	0,5023	19,0223
20	3600	10791	317	1,5624	46,6007	2,4457	0,5345	37,5779
21	4800	14391	320	1,9359	80,0503	3,5401	0,7962	63,2861
22	2200	6591	323	1,4406	19,1185	1,6009	0,3307	13,9734
23	3300	9891	326	1,4245	38,9456	2,1777	0,5216	30,0593
24	2300	6891	327	1,3953	21,0951	1,6057	0,3636	16,3870
25	4000	11991	334	1,6482	63,3897	3,4508	0,5968	44,6902
26	1400	4191	336	1,1118	7,8580	1,1658	0,2612	5,7148
27	3000	8991	348	1,5218	31,4098	2,2385	0,5604	24,8093
28	2900	8691	354	1,5979	31,1404	1,9686	0,4640	23,4944
29	4400	13191	354	2,1528	70,9354	3,5324	0,7528	55,1740
30	3800	11391	356	2,0614	53,5172	3,2058	0,6052	40,7595
31	3500	10491	357	1,7344	44,9894	2,8085	0,6078	34,5449
32	4100	12291	357	2,1478	62,6602	3,8749	1,0054	49,8159
33	2500	7491	361	1,7457	26,1102	2,2222	0,4612	18,9346
34	4500	13491	364	2,1433	73,7658	3,6554	0,8948	56,2307
35	3400	10191	366	1,6623	42,0242	2,3720	0,6446	31,7536
36	4200	12591	374	2,1236	67,6515	3,2935	0,6621	53,1859
37	3200	9591	380	1,8491	39,7250	2,6442	0,7016	28,4207
38	2600	7791	384	1,6175	25,7014	2,1368	0,4619	18,7650
39	3900	11691	384	2,1711	55,9889	3,2844	0,7531	43,0636
40	2700	8091	387	1,9483	28,8155	2,3264	0,5515	22,0799
41	4600	13791	399	2,1501	74,2540	3,5349	0,8618	59,6226
42	4900	14691	399	2,8592	92,6384	4,5817	0,8818	70,8729
43	3100	9291	405	2,2097	35,2753	3,2153	0,5712	26,3412
44	4700	14091	453	2,8120	77,9713	4,2259	0,8083	59,3918
45	4300	12891	468	3,8393	67,9968	4,6346	0,7447	50,4435