Eötvös Loránd Tudományegyetem

Természettudományi Kar

Kristóf Panna

Optimális független fák keresése gráfokban

Szakdolgozat

Alkalmazott matematikus MSc

Számítástudomány szakirány

Témavezetők:

Belső témavezető: Bérczi-Kovács Erika, Operációkutatási Tanszék

Külső témavezető: Tapolcai János, BME Távközlési és Médiainformatikai Tanszék

Budapest, 2016

Tartalomjegyzék

Köszönetnyilvánítás

Szeretném megköszönni a témavezetőimnek azt a rengeteg szakmai és emberi támogatást, amit az évek során és a szakdolgozatom összeállításánál is kaptam: Bérczi-Kovács Erikának, aki életem több pontján, és most is hatalmas türelmével, kedvességével, aprólékos gondosságával átsegített a nehézségeken, és az utolsó pillanatig hatalmas energiákkal segített átgondolni a hiányzó részleteket; és Tapolcai Jánosnak, aki egészen befogadott a kutatócsoportjába, így már a TMIT tanszéken is otthon érzem magam, és mindig van egy inspiráló ötlete vagy egy kedves szava. Nagyon jó élmény volt együtt dolgozni.

Bevezető

A dolgozatom tárgya minél kisebb súlyú független fapár keresése irányítatlan, súlyozott gráfokban. Ennek a problémának valós jelentősége van távközlési hálózatok hibavédelmében. A dolgozat gerincét a [10], a 2015-ös IEEE International Conference on Network Protocols konferenciára beválasztott cikk alkotja, aminek a munkafolyamataiban a témavezetőimmel együtt volt alkalmam részt venni.

Az első fejezetben a probléma bemutatása és az elméleti eredmények áttekintése történik.

A második fejezet a két pont közötti minimális összsúlyú útpárt kereső Suurballe algoritmus általános változatát, egy adott r és minden másik csúcs között egyszerre ilyen útpárt kiszámoló Suurballe-Tarjan algoritmus részletes leírását tartalmazza, Tarjan adatszerkezetének ismertetésével, és a helyesség bizonyításával, saját értelmező megjegyzésekkel és ábrákkal kiegészítve, a bizonyítás lépéseit részletesen tárgyalva. Ez azért is szükséges a dolgozat többi részéhez, mert a 3. fejezetben a minél kisebb súlyú független fapár keresésére adott egyik algoritmus ugyan csak szubrutinként használja a Suurballe algoritmust, de az utak speciális kiolvasása miatt a futásideje a Suurballe által adott útjelzők bizonyos tulajdonságain múlik.

Végül a harmadik fejezetben tényleges algoritmusokat láthatunk: a feladat felírását egészértékű lineáris programként, aminek a megoldása optimális, de ezt megoldani nagyobb gráfokra reménytelenül lassú; és néhány gyors heurisztikát, amik fülfelbontásból és s-t rendezésből képeznek fapárt, és egy részük a Suurballe-Tarjan algoritmus eredményét használja fel.

1. fejezet

Független fák

1.1. A feladat megfogalmazása

1.1.1. Motiváció

A mérnöki gyakolatban felmerülő probléma, hogy hogyan lehet egy internetprotokoll (IP) hálózatban olyan megbízható forgalomirányító algoritmust tervezni, hogy egy vezeték (vagy csomópont) meghibásodása esetén is képesek legyenek kommunikálni a számítógépek. Erre jó megoldás lehet, ha egyszerre több, lehetőleg diszjunkt útvonalon küldjük a csomagokat, vagy hiba esetén átkapcsolunk egy alternatív útvonalra (multipath routing), így nem kell a hiba előfordulásakor megvárni, amíg újra feltérképezzük a hálózatot és újraszámolunk mindent. Ha nem fordul elő hiba, ez a módszer akkor is csökkentheti a késleltetést.

Ilyen algoritmusok már fejlesztés alatt állnak, ám ezek rosszul skálázódnak: nagy hálózatokban a sok különböző útvonal számontartása nagyon sok helyet foglal, például ha minden router számontart minden start- és célcsúcshoz külön-külön k darab útvonalat, akkor $k \cdot n^2$ bejegyzés szükséges (ahol n a csúcsok száma).

Olyan, már alkalmazott továbbító algoritmusra építünk (destination-based hop-by-hop forwarding), ahol routing táblák alapján előre adottak az útvonalak: a hálózat gráfjában egy adott r csomópontba küldött minden csomagot egy fán küldhetünk, azaz elég, ha minden x router tudja, hogy az ebbe az r csomópontba címezett csomagokat melyik $f(x)$ irányba kell továbbküldeni (ezek összesen egy be-fenyőt határoznak meg, praktikusan a célcsúcsra vonatkozó legrövidebb utak fenyőjét). Az ötlet az, hogy k különböző ilyen fát tartunk számon minden célcsúcshoz, melyek így minden startcsúcsból k útvonalat határoznak meg a célba. Így minden routernek elég legfeljebb $k \cdot n$ féle lehetséges továbbítást megjegyezni (ami csak lineáris a csúcsszámban, szemben a fenti négyzetessel). A továbbiakban a $k = 2$ esetet vizsgáljuk.

Tehát a kihívás az, hogy a két fát úgy válasszuk meg, hogy az adott célcsúcsba minden másik csúcsból a fákban futó egy-egy útvonal diszjunkt legyen, de közben egyrészt

ne legyenek az így kapott útvonalak "sokkal" hosszabbak a lehetséges legrövidebb útnál, másrészt a fákat kiszámító algoritmus idő- és tárigénye se haladja meg jelentősen a legrövidebb utak fáját kiszámító algoritmusét. Ez alapján gráfelméleti eszközökkel vizsgálható kérdéseket kapunk.

1.1.2. A feladat gráfokban

Legyen $G = (V, E)$ irányítatlan, egyszerű gráf, $|V| = n$, $|E| = m$.

A műszaki kérdésből a következő definíció következik közvetlenül:

1.1. Definíció (Redundáns fák)**.** Legyen $T_1, T_2, \dots T_k$ k darab fa G-ben, és $r \in V(G)$ egy kijelölt csúcs. Tegyük fel, hogy minden $v \in V$ csúcsból létezik valamelyik fában r-be vezető út. Ekkor azt mondjuk, hogy ezek a fák r **gyökerű redundáns fák**, ha bármelyik élet elhagyva a gráfból továbbra is lesz út minden pontból az r-be legalább az egyik fában. Továbbá **pontredundáns fák**nak nevezzük őket, ha bármely $v \neq r$ pontot (és a ráilleszkedő éleket) elhagyva a gráfból, továbbra is lesz út bármely pontból az r-be legalább az egyik fában. Speciálisan $k = 2$ esetén **(pont)redundáns fapár**ról beszélünk.

Mivel a $k = 2$ esetet fogjuk vizsgálni, tekintsük a következő definíciót:

1.2. Definíció (Független feszítőfák)**.** Legyen T_1 és T_2 két feszítőfa G-ben, és $r \in V(G)$ egy kijelölt csúcs. Azt mondjuk, hogy T_1 és T_2 **r-(él)független**ek, ha G minden r-től különböző v csúcsa esetén a v-ből r-be T_1-ben illetve T_2-ben vezető egyértelmű utak belsőleg (él)diszjunktak.

Például tekintsük az 1.3.1. szakaszban az 1.2d. ábrán látható, pirossal illetve kékkel jelölt független fapárt.

1.3. Állítás. *Legyen T_1 és T_2 két fa G-ben, $r \in V(G)$. Ekkor T_1 és T_2 akkor és csak akkor redundáns fapár, ha élfüggetlen feszítőfapár (és akkor és csak akkor pontredundáns fapár, ha független feszítőfapár).* □

Mivel az sokszor kényelmesebb, a független fapár definíciójával fogunk tovább dolgozni.

1.4. Megjegyzés. Az élfüggetlenség gyengébb feltétel, mintha a feszítőfák éldiszjunktságát követelnénk meg, pl. legyen G egy 3 hosszú kör: ebben létezik élfüggetlen feszítőfapár, de nem létezik éldiszjunkt feszítőfapár.

Hasznos lesz a következő észrevétel:

1.5. Állítás. *Tegyük fel, hogy (T_1, T_2) r-(él)független fapár. Irányítsuk T_1 és T_2 éleit r felé. Ekkor nem létezik olyan uv él, ami T_1-ben is és T_2-ben is szerepel ugyanazzal az irányítással.*

Bizonyítás: Indirekt tegyük fel, hogy $\overrightarrow{uv} \in T_1, T_2$. Ekkor u csúcsból a T_1 és a T_2 fabeli $u \to r$ út első éle is $\overrightarrow{uv}$, mert minden csúcsból csak egy T_1-beli és egy T_2-beli él mutathat kifelé. Tehát az $u \to r$ utak nem diszjunktak, sőt nem is éldiszjunktak, ami ellentmond a fapár (él)függetlenségének. □

1.6. Definíció (Fabeli út a gyökérbe)**.** Adott r gyökerű T feszítőfában bármely v csúcsból egyértelmű út vezet (lefelé) r-be. Ezt a v-r utat jelölje $P(T, v)$.

Ha eltekintünk az útvonalak hosszának lerövidítésétől, akkor már összesen két élfüggetlen fával lehetséges olyan routing protokollt létrehozni minden csúcsból minden csúcsba, ami ellenáll egy élhibának.

1.7. Protokoll (Két fás protokoll élhibára [4])**.** *Legyen (T_1, T_2) egy r-élfüggetlen fapár. Üzenetküldés u csúcsból v csúcsba:*

1. *u elküldi az üzenetet r gyökérnek (lefelé) mindkét fán (ehhez elég minden v csúcsnak a saját, mindkét fabeli szülőjét ismernie);*
2. *mikor a gyökér megkapja az üzenetet, akkor elküldi (felfelé) a $P(T_1, v)$ és a $P(T_2, v)$ úton is v-nek (vagy akár minden csúcsba szétküldi mindkét fán, így senkinek nem kell ismernie a v-be vezető $P(T_i, v)$ utakat, csak a saját gyerekeit a fákban).*

Mivel az 1.1. Definíció szerint feltettük, hogy legfeljebb egy él sérül meg, felfele és lefele haladva is sikeres lesz az üzenet továbbítása legalább az egyik úton, tehát valóban egy él-hibavédelmet biztosít a protokoll.

Persze az, hogy egyetlen csúcson minden üzenet áthalad, egyrészt sérülékennyé teszi a hálózatot, másrészt erősen terheli az r csúcsot, ráadásul az úthosszak minimalizálása esetén szinte biztosan szuboptimális (bár üzenetenként így is csak $O(n)$ darab üzenetküldés történik akkor is, ha "felfelé" az egész fába szétküldjük az üzenetet, minden élen legfeljebb 2 darab). Viszont így általában minden csúcsnak kevesebb információt kell tárolnia, mint az 1.1.1. szakaszban felvázolt módszernél, mikor minden r célcsúcsra külön fapárt tartunk számon.

Mint azt az 1.2.1. szakaszban látni fogjuk, 2-(él)összefüggő gráfban (él)független fapár keresése lehetséges $O(m)$ időben (nem 2-élösszefüggőben pedig nem létezik). A dolgozat fő témája viszont a súlyozott esetben minél jobb fapár megtalálása, ami már bonyolultabb probléma.

1.1.3. Optimalizálási feladatok

Tekintsük a G élein a $c : E \to \mathbb{R}^+$ súlyfüggvényt (az alkalmazásban ez a hálózat fizikai adottságaiból adódik). Ugyan 2-összefüggő gráfban mindig létezik független fapár, de most valamilyen értelemben a "legkisebb súlyú" fapárt szeretnénk megtalálni. Egy fapár súlyát többféle módon lehet definiálni, amik más és más optimalizálási feladatokhoz vezetnek.

1.8. Definíció (Pont fabeli útjának súlya)**.** Egy v csúcsra a T fesztőfában a $P(T, v)$ v-r út éleinek összsúlyát jelölje

$$c_T(v) = \sum_{e \in P(T,v)} c(e).$$

Az 1.1.1. szakaszban beutatott problémában [10] a fő célunk az, hogy a csúcsoktól r-be vezető útpárok hosszai összesen minimálisak legyenek, így a főként vizsgált célfüggvényben ezek összegét tekintjük.

1.9. Definíció (Független fapár összsúlya)**.** Egy (T_1, T_2) fapár összsúlya legyen:

$$c(T_1, T_2) := \sum_{v \in V \setminus \{r\}} (c_{T_1}(v) + c_{T_2}(v)).$$

A fő célunk ezt a $c(T_1, T_2)$ mennyiséget minimalizálni (tehát ahol nem mondjuk ki külön, ott erre optimalizálunk).

Ezen kívül vizsgálunk majd egy másik célfüggvényt is, amihez először bevezetünk még két fogalmat.

1.10. Definíció (Csúcs legrövidebb útpárjának összsúlya)**.** Jelölje $v \in V \setminus \{r\}$ csúcsra

$$d_2(v) := \min\{c(P_1) + c(P_2) : P_1, P_2 \text{ belsőleg diszjunkt } v\text{-}r \text{ utak}\}$$

(ahol $c(P) = \sum_{e \in P} c(e)$).

1.11. Definíció (Eltérés csúcsban a legrövidebb útpárhoz képest)**.** Legyen (T_1, T_2) r-független fapár; ekkor egy $v \in V \setminus \{r\}$ csúcsra legyen

$$Gap_{T_1,T_2}(v) := \frac{c_{T_1}(v) + c_{T_2}(v)}{d_2(v)}.$$

A most következő célfüggvény azt méri, hogy egy csúcs legfeljebb mennyire "járhat rosszul" ezzel a *Gap* értékkel mérve.

1.12. Definíció (Legnagyobb elétérés a legrövidebb útpárhoz képest)**.** Egy fapár útjainak legnagyobb elétérése a legrövidebb útpárhoz képest:

$$MaxGap(T_1, T_2) = \max_{v \in V \setminus \{r\}} Gap_{T_1,T_2}(v).$$

Itt is minimalizálni igyekszünk a célfüggvényt.

A fenti definíciókat pontfüggetlenséggel mondtuk ki, de élfüggetlenséggel természetesen hasónlóan definiálható.

A továbbiakban főleg pontfüggetlen fapárokra és csúcsdiszjunkt utakra koncentrálunk.

1.2. Független fapár keresése fülfelbontással és s-t rendezéssel

1.2.1. Független fapár létezése 2-összefüggő gráfokban

Ebben a szakaszban megmutatjuk, hogy amennyiben a hálózatunk 2-összefüggő (ami szükséges feltétel, lásd az 1.15. Állítást), akkor léteznek független fák, tehát az optimalizálási kérdés értelmes. Az itt bevezetett fogalmak és a bizonyításban használt algoritmusok adják az alapját a 3.2.1. szakaszbeli független fapárt kereső algoritmusainknak.

Először idézzünk fel néhány kulcsfontosságú fogalmat.

1.13. Definíció. Egy G 2-összefüggő gráf **fülfelbontása** olyan $(P_0, P_1, \ldots, P_k)$ sorozat, ahol P_0 egy kör, a P_i $(1 \leq i \leq k)$ pedig olyan u_i-v_i út, melyre $V(P_i) \cap (V(P_0) \cup V(P_1) \cup \cdots \cup V(P_{i-1})) = \{u_i, v_i\}$, és $G = P_0 \cup P_1 \cup \cdots \cup P_k$. **Fül**nek nevezzük a $P_1, \ldots, P_k$ utakat.

Általánosított fülfelbontás esetén megengedjük, hogy a P_i két végpontja egybeessen, míg **nyílt fülfelbontás** esetén megköveteljük, hogy különbözzenek.

A fülfelbontások a következő közismert tétel miatt fontosak.

1.14. Tétel. *Egy gráf akkor és csak akkor 2-élösszefüggő, ha van általánosított fülfelbontása, és akkor és csak akkor 2-összefüggő, ha van nyílt fülfelbontása. Sőt, akárhogyan megválaszthatjuk a P_0 kört, és általában: akármilyen megkezdett $(P_0, \ldots, P_l)$ rész-fülfelbontás kiegészíthető az egész gráf fülfelbontásává.* □

1.15. Állítás. *Ha G gráfban létezik (T_1, T_2) független fapár, akkor G 2-összefüggő.*

Bizonyítás: Tegyük fel, hogy létezik G-ben valamely $r \in V(G)$ gyökérre nézve (T_1, T_2) r-független fapár. Ebből elkészíthetjük G-nek egy nyílt fülfelbontását a következőképpen.

Kezdetben legyen a fedett halmaz $\{r\}$, és amíg amíg van fedetlen csúcs, addig így adunk a fülfelbontáshoz új fület: tekintsük tetszőleges fedetlen $v \in V \backslash \{r\}$ csúcsra a $P(T_1, v)$, $P(T_2, v)$ diszjunkt útpárt, és mindkét útra végezzük el a következőt: v-ből indulva haladjunk az úton, és addig vegyük az út éleit a fülhöz, amíg fedett csúcshoz nem érünk, ez lesz a fül egyik végpontja.

Mivel a fapár pontfüggetlen volt, ezért a $P(T_1, v)$, $P(T_2, v)$ utak pontdiszjunktak, tehát az így kapott fülfelbontás nyílt, ami 1.14. tétel szerint bizonyítja a gráf 2-összefüggőségét. □

A most következő fogalom a másik alapeleme a gondolatmenetnek. Két ekvivalens változatban is megfogalmazzuk.

1.16. Definíció (s-t rendezés)**.** Legyen $G(V, E)$ 2-összefüggő gráf, $st \in E$ él. Egy $\prec$ rendezés a csúcsokon akkor s-t **rendezés**, ha $\forall v \neq s, t$-re:

- $s \prec v \prec t$ (azaz s maximális és t minimális),
- és v-nek léteznek olyan u, w szomszédai, hogy $u \prec v \prec w$.

Ez nyilvánvalóan ekvivalens a következővel:

1.17. Definíció (s-t számozás)**.** Legyen $G(V,E)$ 2-összefüggő gráf, $st \in E$ él. G csúcsainak egy $g : V \to \{1,\dots,n\}$ számozása **s-t számozás**, ha

- g(s)=1 és $g(t) = n$,
- és $\forall v \neq s,t$ csúcsnak léteznek olyan u, w szomszédai, hogy $g(u) < g(v) < g(w)$.

A konkrét alkalmazástól függ, hogy s-t számozást vagy s-t rendezést érdemes használni: az implementációban praktikusabb a számozás (megengedve nem egész értékeket is), viszont a rendezés általánosítható részben rendezéssé, ami ugyan nehezebben kezelhető, de finomabb módszereknek is teret ad, például a 3.2.1. heurisztikában.

1.18. Definíció (s-t részbenrendezés)**.** Legyen $G(V,E)$ 2-összefüggő gráf, $st \in E$ él. Egy $\prec$ részbenrendezés a csúcsokon akkor **s-t részbenrendezés**, ha $\forall v \neq s,t$-re:

- $s \prec v \prec t$,
- és v-nek léteznek olyan u, w szomszédai, hogy $u \prec v \prec w$.

A részbenrendezéssel ekvivalens fogalom a következő.

1.19. Definíció (s-t irányítás)**.** Egy $G(V,E)$ gráf éleinek egy aciklikus irányítása **s-t irányítás**, ha s az egyetlen forrás és t az egyetlen nyelő.

1.20. Állítás. *Egy s-t irányítás megfeleltethető egy s-t részbenrendezésnek.*

Bizonyítás: Legyen $x \prec y \iff$ létezik $x \to y$ itányított út. Az s-t irányításból így definiált reláció valóban reflexív, antiszimmetrikus (mert a gráf aciklikus), és tranzitív.

Másrészt ha $\prec$ részbenrendezés, akkor ha megirányítjuk a gráf éleit úgy, hogy uv él u-ból v-be mutasson, ha $u \prec v$, akkor s-t irányítást kapunk. □

1.21. Megjegyzés. Egy s-t irányítással rendelkező gráf tetszőleges topologikus sorrendje egy s-t rendezésnek felel meg (de vegyük észre, hogy ez ugyanúgy nem feltétlenül egyértelmű, mint a részbenrendezés kiegészítése teljes rendezéssé).

A szakasz fő tétele a következő fontos tétel, amely azt mutatja, hogy 2-összefüggő gráfban bármely gyökérre nézve találhatunk független fapárt. A bizonyítás elemei adják a 3.2.1. szakaszbeli algoritmusok alapját.

1.22. Tétel (Itai és Rodeh [3])**.** *2-összefüggő $G = (V,E)$ gráfban bármely $r \in V$-re létezik r-független feszítőfapár.*

Bizonyítás: Két lépésben bizonyítunk: megmutatjuk, hogy nyílt fülfelbontásból hogyan képezhető s-t rendezés, abból pedig független fapár.

1.23. Lemma. *Ha $G = (V, E)$ 2-összefüggő, $st \in E$ tetszőleges él, akkor létezik G csúcsainak s-t rendezése.*

Bizonyítás: Vegyük G-nek egy olyan $(P_0, P_1, \ldots, P_k)$ fülfelbontását, ahol P_0 alapkör tartalmazza az st élet, azaz $P_0 = (v_0{=}s, v_1, v_2, \ldots, v_{q-1}{=}t, s)$. Legyen a kör mentén $v_{i-1} \prec v_i$ $(i = 1, \ldots q-1)$, ez P_0-on egy s-t rendezést generál.

Innen indukcióval tegyük fel, hogy $(P_0, P_1, \ldots, P_{j-1})$ rész-fülfelbontás csúcsain már adott egy s-t rendezés; ekkor $P_j = (u_0, \ldots, u_m)$ fülre, ha az eddigi rendezésben $u_0 \prec u_m$, és w az u_0-ra rákövetkező elem $\prec$ szerint, akkor legyen $u_0 \prec u_1 \prec \cdots \prec u_{m-1} \prec w$ ($\prec u_m$ vagy $= u_m$), ami már $(P_0, P_1, \ldots, P_j)$ rész-fülfelbontáson generál s-t rendezést (ha pedig $u_m \prec u_0$ akkor hasonlóan, fordítva). □

1.24. Megjegyzés. A lemma bizonyításából nyerhető algoritmus tehát $O(m)$ időben fülfelbontásból s-t rendezést képez.

1.25. Lemma. *Tegyük fel, hogy adott egy 2-összefüggő G gráf s-t rendezése. Ebből ($O(m)$ időben) előállítható egy s-független feszítőfapár.*

Bizonyítás: Irányítsuk meg G éleit úgy, hogy uv él u-ból v-be mutasson, ha $u \prec v$ (ekkor s-t irányítást kapunk), kivéve az st élt, ami pedig t-ből s-be. Az így kapott irányított gráf erősen összefüggő. Keressünk az irányított gráfban egy s-ből kifelé irányított feszítőfenyőt, ez lesz T_1; és egy befelé irányítottat, ez lesz T_2. Így T_1, T_2 az irányítatlan gráfban s-függetlenek, mert egy tetszőleges v csúcsból a T_1-beli út csak olyan u csúcsokon halad keresztül, melyekre $u \prec v$; a T_2-beli pedig csak olyan w csúcsokon, amikre $v \prec w$ (kivéve s-et, de $ts \in T_2$). □

Tehát adott s csúcsra elég egy tetszőleges s-re illeszkedő st élre venni a G csúcsainak egy s-t rendezését, majd egy abból képzett s-független feszítőfapárt. □

A fenti bizonytásból algoritmus is nyerhető, mely egy 2-összefüggő gráf egy adott nyílt fülfelbontásából s-t rendezést/számozást, irányítást, majd független fapárt generál $O(m)$ időben. (Arról még nem beszéltünk, hogy fülfelbontást hogyan állítunk elő, erről a 3.2. szakaszban lesz szó.) A módszer egyszerűsége és gyorsasága miatt a 3.2. szakaszban ismertetett heurisztikák mind erre épülnek.

Az algoritmus egyszerűen átalakítható közvetlenül az élfüggetlen esetre.

1.26. Tétel (Itai és Rodeh [3])**.** *Legyen G egy 2-élösszefüggő gráf, és r a G egy tetszőleges csúcsa. Ekkor G-ben létezik két r-élfüggetlen feszítőfa.*

Bizonyítás: Ha G 2-összefüggő, akkor a pontfüggetlenre vonatkozó algoritmussal készen vagyunk. Ha nem, akkor 2-összefüggő blokkonként haladhatunk hasonlóan a következőképpen.

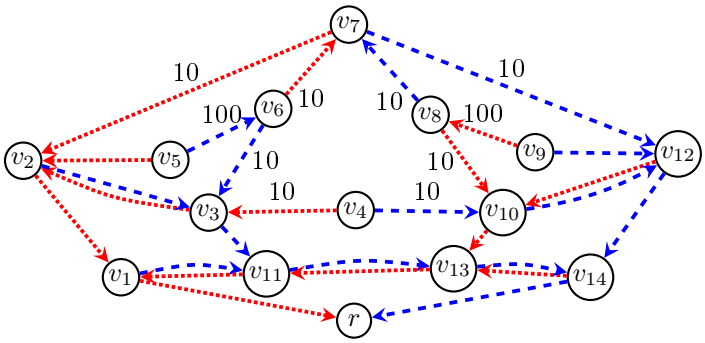
Vegyük a gráf maximális 2-összefüggő blokkokra való $r \in B_0, B_1, \ldots, B_k$ felbontását (ez $O(m)$ időben megtehető). Mivel G 2-élösszefüggő, ezért nem lesznek elvágó élek, és

így a csúcsok által elválasztott blokkok fa szerkezetbe rendeződnek. B_0 blokkot gyökérnek tekintve megirányíthatjuk a blokk-fát B_0-ból, így minden B_i $(i > 0)$ blokknak lesz egy B_j szülője, amihez egy r_i elvágó ponttal csatlakozik. Legyen $r_0 := r$. Legyen minden B_i blokkban (T_1^i, T_2^i) pontfüggetlen feszítőfapár (amit a fentiek alapján tudunk keresni). Ezek összeillesztéséből előállíthatók a $T_1 = \bigcup_{i=0}^{m} T_1^i$ és a $T_2 = \bigcup_{i=0}^{m} T_2^i$ fák, amik r-élfüggetlen feszítőfák lesznek.

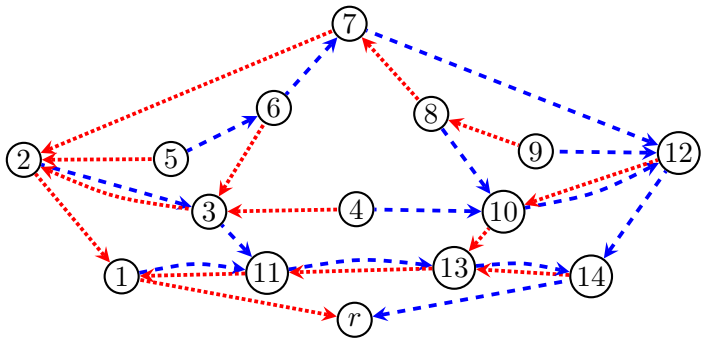
□

1.2.2. A fülfelbontásos módszer szuboptimalitása

Ebben a szakaszban eddig csak független fapár létezésével foglalkoztunk súlyozatlan esetben, fontos azonban megjegyezni, hogy a súlyozott feladatban optimális fapár nem mindig áll elő fülfelbontásból (ill. s-t rendezésből), tekintsük például az 1.1. ábrán a gráfot.



(a) Optimális fapár. A csúcsokhoz tartozó fabeli útpár minden csúcsban a legrövidebb.



(b) A legjobb fülfelbontásból előálló fapár (a csúcsokra írt s-t számozás generálja). v_{10} és v_6 csúcsok fabeli útpárja 10-zel hosszabb, mint a csúcsok legrövidebb útpárjai.

1.1. ábra. Példa arra, hogy az optimális fapár nem mindig áll elő fülfelbontásból. Az üres élek költsége 0. [10]

1.27. Állítás. *Az 1.1. ábrán látgató gráfban egyik optimális fapár sem áll elő fülfelbontásból.*

Bizonyítás: Az 1.1a. ábrán látható fapár optimális, mert minden v csúcshoz olyan útpár tartozik a fában, aminek az összsúlya minimális a gráfban. Sőt, ahogy mindjárt meg-

mutatjuk, lényegében az egyértelmű ilyen. Viszont nem áll elő fülfelbontásból, hiszen $(v_7, v_8, v_{10}, v_4, v_3, v_6)$ kör a fákat generáló s-t irányításban irányított kör lenne, de egy s-t irányítás csak aciklikus lehet, mert s-t rendezéshez tartozik.

Most bebizonyítjuk, hogy minden optimális fapárban ugyanez a gond előfordul a $(v_7, v_8, v_{10}, v_4, v_3, v_6)$ körrel. Ezt úgy fogjuk belátni, hogy az egyes csúcsokra megvizsgáljuk, hol lehet a gráfban a legrövidebb útpárjuk r-be, és vesszük azokat az éleket, amiken minden lehetséges legrövidebb útpárban benne vannak (figyelembe véve a már rögzített faéleket is). Ezekből az információkból felépítjük a fapárt, kihasználva az 1.5. Állítást.

1. Először vegyük a v_1 csúcsot. Ennek a legrövidebb útpárja csupa 0 éleken vezet, amik többféleképpen is elhelyezkedhetnek, ezek közül a közös: nevezzük pirosnak azt a fát, ami a $\overrightarrow{v_1 r}$ élet tartalmazza (a másikat kéknek); ekkor a $\overrightarrow{v_{14} r}$ él a kék fához tartozik; a $\overrightarrow{v_{11} v_{13}}$ piros és a fordítottja kék.

2. A v_2 útpárjai is 0 költségűek, itt az előzőket figyelembe véve a fix élek: a $\overrightarrow{v_2 v_1}$ piros; a $\overrightarrow{v_3 v_2}$ piros és a fordítottja kék; a $\overrightarrow{v_3 v_{11}}$ kék.

3. A v_{12} csúcsnál hasonlóan: a $\overrightarrow{v_{12} v_{14}}$ kék; a $\overrightarrow{v_{12} v_{10}}$ kék és a fordítottja piros; a $\overrightarrow{v_{10} v_{13}}$ piros.

4. Most tekintsük a v_4 csúcs 20 összhosszú útjait. A fentiekben rögzített élek miatt a $\overrightarrow{v_4 v_3}$ csak piros, és a $\overrightarrow{v_4 v_{10}}$ csak kék lehet.

5. Következik a v_5, itt az útpár 110 hosszú lesz, tehát a v_6 felé induló út nem érintheti a v_7 csúcsot, tehát csak v_3 felé haladhat tovább; a v_2 felé induló út tehát csak a v_1 felé. Így az előzők miatt a $\overrightarrow{v_5 v_2}$ piros; a $\overrightarrow{v_5 v_6}$ kék; és a $\overrightarrow{v_6 v_3}$ kék.

6. Hasonlóan a v_9 csúcsnál: a $\overrightarrow{v_9 v_{12}}$ kék; a $\overrightarrow{v_9 v_8}$ piros; és a $\overrightarrow{v_8 v_{10}}$ is piros.

7. A v_7 csúcsnál a minimális útpár összsúlya 20, tehát az utak nem tartalmazhatják a v_6 és a v_8 csúcsokat, így csak a v_2 és a v_{12} felé mehetnek. Már tudjuk, hogy a $\overrightarrow{v_{13} v_{11}}$ piros és a fordítottja kék. Viszont nem lehet, hogy a piros út a $v_7 \to v_{12} \to v_{10} \to v_{13} \to v_{11}$ útvonalon indul, mert akkor a kék a $v_7 \to v_2 \to v_3 \to v_{11} \to v_{13}$ felé indulna, tehát nem lennének éldiszjunktak. Tehát a $\overrightarrow{v_7 v_2}$ piros és a $\overrightarrow{v_7 v_{12}}$ kék.

8. A v_6 legrövidebb útpárja 30 összsúlyú, tehát egyik út sem vezethet a v_5 felé. Mivel tudjuk, hogy a $\overrightarrow{v_6 v_3}$ kék, ezért a $\overrightarrow{v_6 v_7}$ piros lesz.

9. Hasonlóan a v_8 csúcsnál: a $\overrightarrow{v_8 v_7}$ kék lesz.

Tehát a $(v_7, v_8, v_{10}, v_4, v_3, v_6)$ kör élei az ábrán látható módon lesznek a fákban, tehát akárhogyan folytatjuk a fapár építését a maradék (0 súlyú) éleken, az eredmény nem fog előállni fülfelbontásból. □

Összehasonlításképpen tekintsük az 1.1b. ábra (nem optimális) fapárját, ami pedig előáll fülfelbontásból és s-t rendezésből, ahogy azt az ábrán lévő s-t számozás mutatja; sőt belátható, hogy a legkisebb költségű ilyen fapár.

1.3. Az optimalizálási feladatok tulajdonságai

Az első két alszakasz a $c(T_1, T_2)$ célfüggvénnyel, a harmadik a $MaxGap(T_1, T_2)$ célfüggvénnyel foglalkozik.

1.3.1. Egy alsó korlát az optimumra

A legrövidebb útpárok összsúlyából alsó becslést kaphatunk $c(T_1, T_2)$-re.

1.28. Állítás.

$$\sum_{v \in V \setminus \{r\}} d_2(v) \leq \sum_{v \in V \setminus \{r\}} (c_{T_1}(v) + c_{T_2}(v)) = c(T_1, T_2).$$

□

Ez olyan alsó korlátot ad az optimumra, aminek az értéke $O(m \log_{(1+\frac{m}{n})} n)$ időben számolható a 2. fejezetben bemutatásra kerülő Suurballe-Tarjan algoritmussal [9].

Ez a becslés lehet éles, azaz vannak olyan gráfok, ahol tudunk mutatni olyan független T_1, T_2 fákat, melyek tartalmazzák a csúcsokból r-be futó minimális összköltségű diszjunkt útpárokat (például az 1.1a. ábrán). Ez viszont nem mindig teljesül, sőt (mint az a későbbi NP-teljességi bizonyításból kiderül) annak eldöntése, hogy egy gráf ilyen-e, NP-teljes. Sőt, ezen a fák számának növelése sem segítene: minden $k \in \mathbb{N}$-re létezik olyan gráf, hogy a legrövidebb útpárok nem fedhetőek le k darab redundáns fával (lásd Fehér Dániel példáit ebben: [2]).

Ezért érdemes vizsgálnunk, hogy ez a becslés általában mennyire közelíti jól az optimális független fapár költségét, azaz mennyire jó cél egyáltalán optimális független fákat keresni.

1.29. Definíció (Átlagos úthossz eltérése a legrövidebb útpárok átlagához képest)**.** Jelölje $G = (V, E)$-re, $r \in V$ gyökérre legyen

$$Gap(G, r) := \frac{c(T_1, T_2)}{\sum_{v \in V \setminus \{r\}} d_2(v)}$$

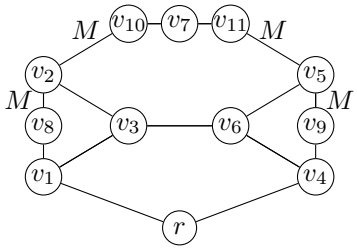
ahol a (T_1, T_2) optimális független feszítőfapár.

A $Gap(G, r)$ érték tehát a gráf tulajdonsága, és azt jelenti, hogy a fenti alsó becslésnek hányszorosa lesz az optimális fapár értéke.

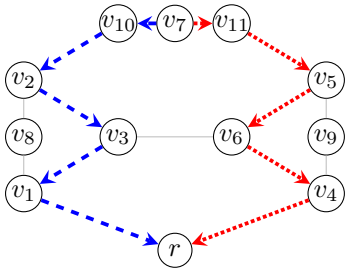
1.30. Sejtés. *$Gap(G, r)$ nem lehet nagyobb $\frac{5}{3}$-nál.*

Sokféle gráfon (valós hálózatokon (103 csúcsig), jellegzetes elméleti példákon és véletlennel generált gráfokon (400 csúcsig)) végzett kísérletek alapján [10] ez az arány mindig $\frac{5}{3} \approx 1{,}66$ alatt maradt, sőt az 1,34 értéket sem haladta meg.

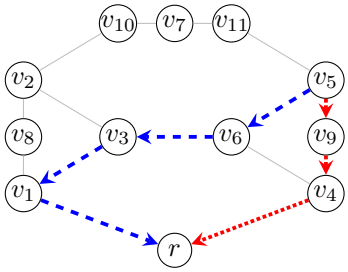
Az $\frac{5}{3}$-os arány közelítésére az 1.2. ábrán látható példa, ahogy azt a következő állítás mutatja.



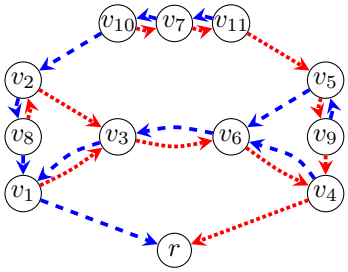
(a) A G_M gráf (egységköltséggel, az M-mel jelölt élek egy-egy M hosszú utat jelölnek).



(b) A v_7 legrövidebb útpárja.



(c) A v_5 legrövidebb útpárja.



(d) Optimális független fapár.

1.2. ábra. Példa olyan gráfcsaládra, ahol a csúcsszám növelésével $Gap(G_M, r) \to \frac{5}{3}$. [10]

1.31. Állítás. *Jelölje G_M azt a gráfot, melyben az 1.2. ábrán az M-mel jelölt éleket kicseréljük egy-egy M hosszú útra. Ekkor az M növelésével*

$$\lim_{M \to \infty} Gap(G_M, r) = \frac{5}{3}.$$

Bizonyítás:

Az r-be vezető legrövidebb útpár összhossza a v_1, v_3, v_6, v_4 csúcsokból 5, a v_8-v_2 út és a v_9-v_5 út csúcsaiból (összesen $2M$ darab) $M+6$, a v_2-v_{10} út és a v_{11}-v_5 út többi csúcsából és a v_7-ből (összesen $2M-1$ darab, ezeket ezentúl nevezzük *felső csúcsok*nak) pedig $2M+8$. Ezek összege $12M^2 + 30M + 36$.

1.32. Állítás. *Az 1.2d. ábrán látható piros (T_1) és kék (T_2) fák optimálisak.*

Bizonyítás: Indirekt tegyük fel, hogy van olyan (F_1, F_2) r-független feszítőfapár, amire $c(F_1, F_2) < c(T_1, T_2)$. Feltehetjük, hogy a $\overrightarrow{v_1 r}$ él kék, így a $\overrightarrow{v_4 r}$ piros. Figyeljük meg, hogy ekkor a kék fa a felső csúcsokat csak a v_2-n keresztül érheti el, mert ha a v_5-ön keresztül érné el őket, akkor (a $\overrightarrow{v_4 r}$ pirossága miatt) a $v_{11} \to v_5 \to v_6 \to v_3 \to v_1 \to r$ végig kék lenne, így a kék fa elvágná a piros fa útját a felső csúcsok felé.

Mivel a (T_1, T_2)-ben csak a felső csúcsok fabeli útpárja hosszabb a G_M-beli legrövidebbnél, ezért az (F_1, F_2)-ben legalább az egyiknek rövidebb útpárja van. Végiggondolható, hogy az egyetlen lehetséges alternatív út a $v_{11} \to v_7 \to v_{10} \to \cdots \to v_2 \to v_3 \to v_1 \to r$, amivel összesen $(2M-1)(M-1) = 2M^2 - 3M + 1$-el csökken a felső csúcsok útpárjainak összköltsége a (T_1, T_2)-belihez képest. Ekkor viszont a v_2-v_8 út csúcsaiból a piros fában csak a felső csúcsokon keresztül vezethet az út (és hasonlóan a v_5-v_9 út csúcsaihoz a kék fában), ami már összesen legalább $(2M-2)(3M+7-(M+6)) = 4M^2 - 2M - 2$-vel növeli a fák összsúlyát, ami nagyobb, mint a csökkenés mértéke, tehát $c(F_1, F_2) > c(T_1, T_2)$, ellentmondás. Tehát a (T_1, T_2) valóban optimális. □

$c(T_1, T_2) = 20M^2 + 26M + 32$, tehát

$$\lim_{M\to\infty} Gap(G_M, r) = \lim_{M\to\infty} \frac{20M^2 + 26M + 32}{12M^2 + 30M + 36} = \frac{5}{3}.$$

□

1.3.2. NP-teljesség

Az NP-teljesség bizonyításához [10] a következő alakban célszerű felírni a feladatot:

1.33. Definíció (Minimum Length Redundant Trees (MLRT) probléma)**.** Adott $G(V, E)$, $c : E \to \mathbb{R}^+$ élsúly, r csúcs, $k > 0$. Létezik-e olyan (T_1, T_2) r-(él)független fapár, melyre $c(T_1, T_2) \leq k$?

1.34. Tétel. *Az MLRT eldöntése NP-teljes (az él- és a pontverzió is).*

Bizonyítás: Az MLRT probléma NP-beli, mert egy megfelelő (T_1, T_2) fapár polinomiális időben ellenőrizhető tanú (hiszen $c(T_1, T_2)$ kiszámolása és a függetlenség ellenőrzése is megtehető $O(mn)$ időben). Tehát elég látnunk, hogy NP-nehéz, ehhez visszavezetjük rá a következő ismert NP-teljes problémát:

1.35. Definíció (NAE-3SAT probléma)**.** A 3SAT-hoz hasonlóan a Not-All-Equal 3-Satisfiability problémában minden klóz pontosan 3 literált tartalamaz, viszont itt csak azt tekintjük kielégítő megoldásnak, ahol minden klóz tartalmaz *igaz és hamis* literált is.

Legyen (X, C) egy NAE-3SAT példány, ahol $X = \{x_1, \ldots, x_n\}$ a változók és $C = \{c_1, \ldots, c_n\}$ a klózok halmaza. Ehhez készítsünk egy $G = (V, E)$ gráfot, ahol:

- a csúcsok:
  - r gyökér;
  - minden $x_i \in X$ változóhoz: x_t^i, x_f^i (amik az *igaz* (t) illetve *hamis* (f) értékadásnak felelnek meg);
  - minden $c_j \in C$ klózhoz: c^j;

- az élek:
  - minden $x_i \in X$ változóhoz: x_t^i, x_f^i csúcsokat összekötjük r-rel és egymással;
  - minden $c_j \in C$ klózhoz:
    - ha $x_i \in c_j$, akkor (x_t^i, c_j) lesz él;
    - ha $\bar{x}_i \in c_j$, akkor (x_f^i, c_j) lesz él.

(A költség minden élen legyen 1.)

1.36. Megjegyzés. Így a $v = x_t^i$ illetve x_f^i típusú csúcsokra $d_2(v) = 3$, a $v = c^j$ típusúakra pedig $d_2(v) = 4$.

1.37. Állítás. *Akkor és csak akkor létezik megoldása az (X, C) NAE-3SAT példánynak, ha G-ben van olyan (optimális) (T_1, T_2) fapár, ahol $c(T_1, T_2) = \sum_{v \neq r} d_2(v)$ (azaz $Gap(G) = 1$).*

1.38. Megjegyzés. Itt tehát a az MLRT definíciójában megjelenő $k = \sum_{v \neq r} d_2(v)$ (ami alsó becslés).

Bizonyítás: Legyen $a : X \to \{t, f\}$ egy jó értékadás (X, C)-hez, $\bar{a}$ pedig az ellenkezője (ahol $\bar{a}(x_i) = t \iff a(x_i) = f$).

Jelöljön minden $c_i \in C$ klózra $x_{t(i)}$ egy olyan literált c_i-ben, aminek az értéke igaz lesz a értékadás mellett (azaz vagy $x_{t(i)} \in c_i$ és $a(x_{t(i)}) = t$, vagy $\bar{x}_{t(i)} \in c_i$ és $a(x_{t(i)}) = f$), és hasonlóan $x_{f(i)}$ egy olyan literált c_i-ben, ami hamis.

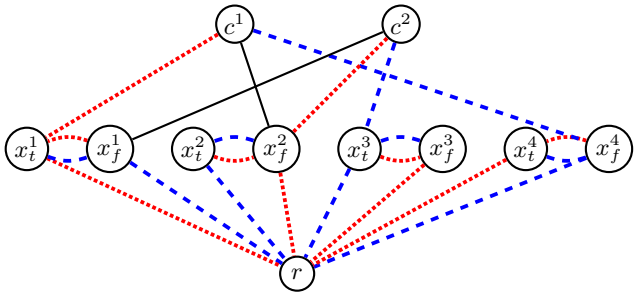
Ezek alapján készítsük el T_1, T_2 fákat:

$$T_1 = \{(x^j_{a(x_j)}, r), (x^j_{\bar{a}(x_j)}, x^j_{a(x_j)}) | x_j \in X\} \cup \{(c^i, x^{t(i)}_{a(x_{t(i)})}) | c_i \in C\}$$

$$T_2 = \{(x^j_{\bar{a}(x_j)}, r), (x^j_{a(x_j)}, x^j_{\bar{a}(x_j)}) | x_j \in X\} \cup \{(c^i, x^{f(i)}_{\bar{a}(x_{f(i)})}) | c_i \in C\}$$

(ahol tehát $x^j_{a(x_j)}$ az x_j változó a szerinti értékétől függően x_t^j vagy x_f^j, az $x^{t(i)}_{a(x_{t(i)})}$ pedig a c_i klózban egy a szerint igaz literálhoz tartozó x_q változó a-beli értéke szerint x_t^q vagy x_f^q (és $x^{f(i)}_{\bar{a}(x_{f(i)})}$ hasonlóan a c_i klózban egy a szerint hamis literálhoz tartozó x_q változó a-beli értékének ellenkezője szerint x_f^q vagy x_t^q)).

Tehát a T_1 tartozik az a értékadás szerint igaz literálokhoz, a T_2 pedig a hamisakhoz.



1.3. ábra. ($X = \{x_1, x_2, x_3, x_4\}, C = \{c_1 = \{x_1, \overline{x}_2, \overline{x}_4\}, c_2 = \{\overline{x}_1, \overline{x}_2, x_3\}\}$) NAE-3SAT feladathoz tartozó gráf. Egy jó értékadás $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1$; az ábrán az ehhez tartozó fákat látjuk. [10]

Ezek a fák függetlenek és minden csúcsra a legrövidebb útpárt tartalmazzák, mert:

- c_i-ből r-be:

  $$\{(c_i, x^{t(i)}_{a(x_{t(i)})}), (x^{t(i)}_{a(x_{t(i)})}, r)\} = P(T_1, c_i)$$

  és

  $$\{(c_i, x^{f(i)}_{\bar{a}(x_{f(i)})}), (x^{f(i)}_{\bar{a}(x_{f(i)})}, r)\} = P(T_2, c_i)$$

  legrövidebb (4 hosszú) útpár, és diszjunktak, mert $x^{t(i)}_{a(x_{t(i)})} \neq x^{f(i)}_{\bar{a}(x_{f(i)})}$ (mert ha $t(i) \neq f(i)$, akkor a csúcsok felső indexe különbözik, ha pedig $t(i) = f(i)$, akkor $\bar{a}(x_{f(i)}) \neq a(x_{t(i)})$), és $t(i)$, $f(i)$ léteznek minden i-re, mert a jó értékadás;

- x_t^i, x_f^i csúcsokból r-be pedig a megfelelő 3 hosszú útpárokat tartalmazzák a fák.

A másik irányhoz legyen (T_1, T_2) olyan független fapár, amire T_1, T_2 tartalmazzák minden csúcsra az egyértelmű legrövidebb útpár egy-egy útját (spec. x_f^i, x_t^i csúcsokra is). Így minden $x_i \in X$-re az $\{(x_t^i, x_f^i), (x_f^i, r)\}$ $x_t^i r$-utat tartalamazza az egyik fa, míg az $\{(x_f^i, x_t^i), (x_t^i, r)\}$ $x_f^i r$-utat a másik (hiszen (x_t^i, x_f^i) élen egy adott fabeli utak csak az egyik rányba mehetnek).

Tehát értelmes a következő értékadás:

$$a(x_i) := \begin{cases} t & \text{ha } (x_t^i, r) \in T_1 \\ f & \text{ha } (x_f^i, r) \in T_1 \end{cases}$$

A c_i csúcs legrövidebb útpárjának tartalmazása miatt létezik olyan x_j változó, amire vagy $x_j \in c_i$ és $(x_t^j, r) \in T_1$, vagy $\bar{x}_j \in c_i$ és $(x_f^j, r) \in T_1$ (mivel a c_i csúcs T_1-beli útja vagy x_t^j vagy x_f^j típusú csúcson keresztül vezet). Tehát van olyan literál c_i-ben, aminek az értéke *igaz*. Hasonlóképpen a c_i másik útját pedig tartalmazza T_2, tehát van olyan literál c_i-ben, aminek az értéke *hamis*. Tehát a egy jó értékadás lesz a NAE-3SAT feladathoz, így a másik irányt is beláttuk.

□

Tehát mivel a NAE-3SAT NP-teljes, ezért az MLRT valóban NP-nehéz, tehát NP-teljes is. □

1.3.3. Maximális eltérés egy csúcsnál a legrövidebb útpárhoz képest

Érdekes kérdés, hogy a $MaxGap(T_1, T_2)$ érték legfeljebb mekkora lehet az erre nézve optimális fapárban.

Nyilván $\min_{T_1,T_2 \text{ független}} MaxGap(T_1, T_2) \geq Gap(G, r)$.

1.39. Sejtés. *Az optimális fapárban* $MaxGap(T_1, T_2) \leq 2$.

Ennek a célfüggvénynek az optimalizálására ugyan nem készült algoritmus, de a mérések során még a másik $c(T_1, T_2)$ célfüggvényre nézve optimális fapárban sem haladta meg a $MaxGap$ értéke a 0,25-öt sem.

A $MaxGap(T_1, T_2) = 2$ érték közelítésére tekintsük az 1.4. ábrán látható példát.

1.40. Állítás. *Az 1.4. ábrán látható* G_k *gráfban a* v_iv_{i+1} *él súlya* 2^i*, a jelöletlen éleken pedig 0 a súly. A pontok számát növelve* G_k*-ban*

$$\lim_{k\to\infty} \min_{T_1,T_2 \text{ } \textit{függetlenek}} MaxGap(T_1, T_2) = 2.$$

Bizonyítás: Tegyük fel, hogy létezik r-független fapárok egy olyan (T_1^k, T_2^k) családja, hogy G_k-ban minden v csúcsra $Gap_{T_1,T_2}(v) < 2$. Belátjuk, hogy ez csak egyféleképpen jöhet létre, amikor viszont v_k csúcsra $\lim_{k\to\infty} Gap_{T_1,T_2}(v_k) = 2$.

Minden u_i-re $d_2(u_i) = 0$, tehát az u_i csúcsok legrövidebb útpárjainak egyértelműsége miatt rv_0, v_0u_1, u_1u_2, $\ldots u_{k-1}u_k$, u_kr éleken a fák csak az 1.4b. ábrán látható módon helyezkedhetnek el. (Legyen a v_kr élet tartalmazó T_1 fa a piros, a v_0r-et tartalmazó T_2 pedig a kék.)

Vizsgáljuk meg sorban az u_iv_i éleket. Minden ilyen élen legfeljebb az egyik fa haladhat át a gyökér felé $\overrightarrow{v_iu_i}$ irányban. Mivel v_1 csúcs legrövidebb útpárja 1 összsúlyú, és a feltevésünk szerint nem lehet a fabeli útpár kétszerese a legrövidebb útpár összsúlyának, ezért a v_1 piros fabeli útja nem tartalmazhatja a v_1v_2 élt, tehát a $\overrightarrow{v_1u_1}$ élnek pirosnak kell lennie. Hasonlóan indukcióval: tegyük fel, hogy minden $j < i$-re a $\overrightarrow{v_ju_j}$ él piros; mivel v_i legrövidebb útpárja 2^{i-1} összsúlyú, tehát v_i piros fabeli útja nem tartalmazhatja a 2^i súlyú v_iv_{i+1} élt, tehát $\overrightarrow{v_iu_i}$ él is csak piros lehet. Így megkapjuk az egész piros fát, amiből következik, hogy a kék fa csak az 1.4b. ábrán látható lehet.

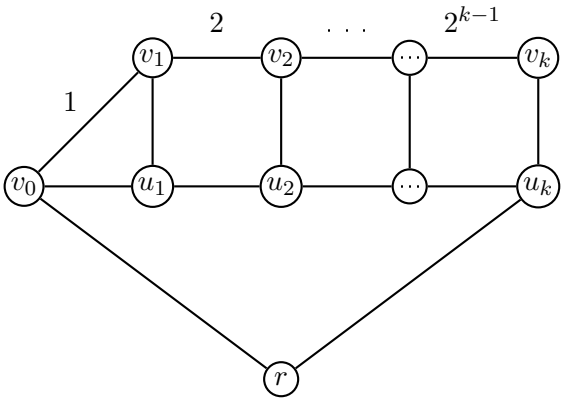
Az így kapott fapárban

$$c_{T_1}(v_k) + c_{T_2}(v_k) = \sum_{i=0}^{k-1} 2^i = 2^k - 1$$

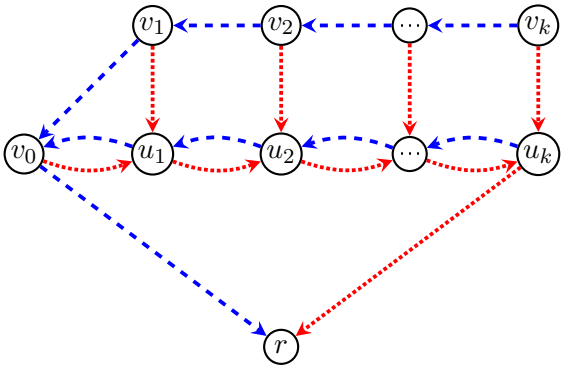
és a v_k csúcs legrövidebb útpárjának összsúlya G_k-ban 2^{k-1} tehát

$$\lim_{k\to\infty} Gap_{T_1,T_2}(v_k) = 2.$$

□



(a) "Lépcső" gráf 2^i súlyozással



(b) $\lim_{k\to\infty} Gap_{T_1,T_2}(v_k) = 2$

1.4. ábra. Olyan G_k gráf, ahol $\lim_{k\to\infty} MaxGap(T_1, T_2) = 2$.

2. fejezet

Legrövidebb útpárok keresése a Suurballe-Tarjan algoritmussal

Mivel a heurisztikák egy része az r-be futó legrövidebb útpárok alapján építkezik, érdemes felidézni az őket megtaláló algoritmust.

Először irányított gráfban, nemnegatív élköltségre, éldiszjunkt útpárokra tekintjük a feladatot, majd azt is látni fogjuk a 2.4. szakaszban, hogyan alakítható át az algoritmus az irányítatlan, konzervatív élköltségű, illetve a pontdiszjunkt esetre. Ez a szakasz nagyrészt a [9] cikk felépítését követi Király Zoltán előadásainak [5] segítségével átdolgozva, és saját magyarázatokkal kiegészítve, de a jelölésekben a dolgozat többi részéhez igazodik.

A cél tehát a következő lesz:

1. Probléma. Adott $G(V, E)$ erősen összefüggő irányított gráf nemnegatív $c : E \to \mathbb{R}_0^+$ élsúlyokkal, és $r \in V$ gyökér ($n = |V|$, $m = |E|$). Keressünk minden $t \neq r$ csúcsra két éldiszjunkt utat r-ből t-be úgy, hogy páronként az összsúlyuk minimális legyen.

Suurballe algoritmusa Tarjan adatszerkezeteivel implementálva ezt a feladatot $O(D)$ időben és $O(m)$ tárral megoldja, ahol $D = m \log_{(1+m/n)} n$ a Dijkstra algoritmus futásidejét jelöli d-edfokú kupaccal implementálva.

Jelölje $d(a, b)$ a legrövidebb $a \to b$ út hosszát G-ben.

2.1. Előkészítés

Az algoritmus mélyebb megértése és a helyesség bizonyítása érdekében először tekintsük egy darab t célcsúcsra a feladatot:

2. Probléma. Adott $G(V, E)$ erősen összefüggő irányított gráf nemnegatív $c : E \to \mathbb{R}_0^+$ élsúlyokkal, és $r, t \in V$. Keressünk A, B éldiszjunkt r-t utakat, melyekre $c(A) + c(B)$ minimális.

A következő algoritmusról könnyen belátható, hogy az optimális megoldást állítja elő erre az egyszerűbb feladatra (lásd 2.2. Állítást).

2.1. Algoritmus (Egyetlen csúcsra A, B útpár megtalálása javítóúttal - első változat)**.**

1. Keresünk A' legrövidebb utat r-ből t-be (Dijkstra);
2. G_t *segédgráf:* az A' éleit megfordítjuk és a költséget negáljuk rajtuk;
3. a G_t-ben keresünk egy B' legrövidebb utat r-ből t-be;
4. az A' és a B' szimmetrikus differenciájából (azaz a szembemenő élpárokat elhagyva $A' \cup B'$-ből) kapjuk $A \cup B$-t.

2.2. Állítás. *A 2.1. Algoritmus optimális megoldást ad a 2. Problémára.*

Bizonyítás: A 2. Probléma a minimális költségű folyam feladat speciális esete (ahol adott az éleken a kapacitás mellett még egy súlyfüggvény is, és a feladat megadni a hálózaton egy x pontból y-ba irányuló, adott k nagyságú folyamot úgy, hogy a folyam költsége minimális legyen). Ugyanis legyen a kapacitásfüggvény értéke G minden élén 1, a súlyfüggvény c, $x := r$, $y := t$; ebben a hálózatban a minimális költségű $k := 2$ értékű folyam éppen az általunk keresett útpárt jelöli ki.

Minimális költségű folyamként tekintve a feladatot, a 2.1. Algoritmus éppen a javító utas algoritmusnak[1] felel meg, ami optimális megoldást ad. □

Mivel a Dijkstra algoritmus csak nemnegatív költségeken működik, ezért módosítsuk úgy a 2.1. Algoritmust, hogy a 2. lépésben negálás helyett inkább *potenciál-eltolást* végzünk: a $\pi(v)$ megengedett potenciál minden pontban legyen $d(r, v)$ (amit az 1. lépésben futtatott Dijkstra algoritmus megadott), és

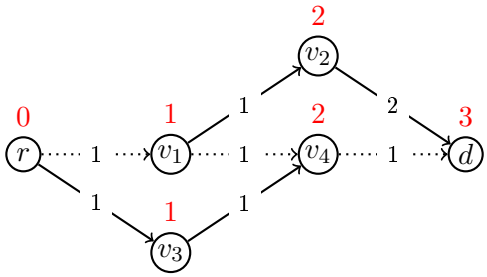
$$c'(uv) := c(uv) + \pi(u) - \pi(v)$$

legyen az új költségfüggvény, így a költség mindenhol nemnegatív. Így a 3. lépésben is alkalmazhatjuk a Dijkstra algoritmust.

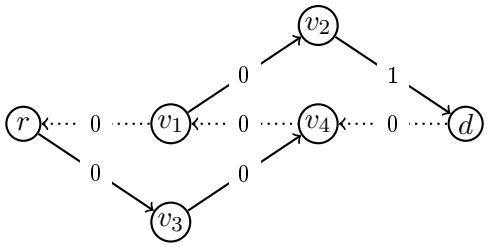
Tehát az egyszerű algoritmusunk végleges változata:

2.3. Algoritmus (Egyetlen csúcsra A, B útpár megtalálása javítóúttal - végleges változat)**.**

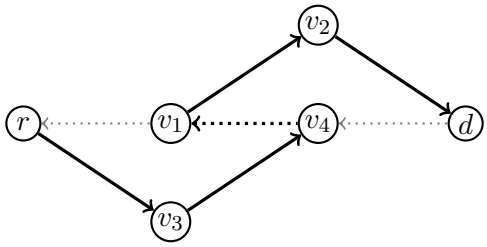
1. Keresünk r-ből minden csúcsba egy legrövidebb utat (Dijkstra); a t-be vezető legrövidebb utat nevezzük A'-nek;
2. G_t *segédgráf:* az A' éleit megfordítjuk, $c'(uv) := c(uv) + d(r, u) - d(r, v)$;
3. a G_t-ben keresünk B' legrövidebb utat r-ből t-be (Dijkstra);
4. az A' és a B' szimmetrikus differenciájából kapjuk $A \cup B$-t.



(a) 1. lépés: Keressünk legrövidebb utat minden csúcsba, a d-be vezető: $A' = r \to v_1 \to v_4 \to d$. Az éleken eredeti c élsúlyok láthatók, minden v csúcs mellett pedig a $d(r, v)$ érték.



(b) 2. lépés: G_d segédgráf: fordítsunk meg minden élet az A' út mentén, és módosítsuk az élsúlyokat a gráfban: $c'(uv) := c(uv) + d(r, u) - d(r, v)$.



(c) 3. lépés: Keressünk G_d-ben legrövidebb utat d-be (ez lesz a *javítóút*): $B' = r \to v_3 \to v_4 \to v_1 \to v_2 \to d$.

2.1. ábra. A 2.3. Algoritmus futása egy d célcsúcsra. A legrövidebb útpár A' és B' szimmetrikus differenciájából: $A = r \to v_1 \to v_2 \to d$ és $B = r \to v_3 \to v_4 \to d$.

Innentől G_t alatt mindig a 2.3. Algoritmusban definiált segédgráfot értjük. Jelölje $d_t(u, v)$ a legrövidebb u-v út hosszát a G_t gráfban.

2.4. Állítás. *A 2.3. Algoritmus optimális megoldást talál.*

Bizonyítás: Egyrészt megfigyelhetjük, hogy egy tetszőleges P v-w útra: $c'(P) = c(P) - d(r, w) + d(r, v)$. Tehát minden adott v, w-re minden v kezdőpontú és w végpontú út költsége ugyanannyival ($d(r, v) - d(r, w)$-vel) változik a potenciál-eltoláskor, tehát két adott pont között a c szerinti legrövidebb út c' szerint is a legrövidebb marad.

Másrészt a potenciál-eltolás után az r-ből induló legrövidebb utak fájának minden élén (azaz A' legrövidebb út élein is) 0 lesz az új költség, tehát az A' élein a c' költségek negáltja

[1] A minimális költségű folyamokra vonatkozó javító utas algoritmus megtalálható például ebben: [6].

megegyezik c'-vel.

Tehát a 2.3. Algoritmus ugyanazt a megoldást találja meg, mint a 2.1. Algoritmus, ami a 2.2. Állítás szerint optimális, azaz a 2.3. Algoritmus is optimális megoldást talál. □

2.5. Megjegyzés. Mivel minden r-t út költsége $d(r,t)$-vel csökken a potenciál-eltoláskor, ezért $d_2(t) := c(A) + c(B) = d_t(t) + 2d(r,t)$.

Vessünk még egy pillantást a 2.3. Algoritmus futásidejére. 1. lépés: $O(D)$, 2. lépés: $O(m)$, 3. lépés: $O(D)$, 4. lépés: $O(n)$. Tehát összesen $O(D)$ idejű.

2.2. Suurballe algoritmusa

Suurballe észrevette, hogy az összes t célcsúcsra egyszerre kiszámolhatjuk az útpárokat $O(D)$ időben (és $O(m)$ tárral), egyetlen Dijkstra-szerű algoritmussal. Az algoritmus hatékonyságának fontos eleme a Tajan által kifejlesztett adatszerkezet, amit később ismertetünk a 2.3. szakaszban. Csúcsonként megvizsgálva a most következő algoritmus futását, a 2.3. Algoritmushoz hasonló működést tapasztalhatunk, ahogy azt a 2.11. Megjegyzésben és a 2.16. Állítás bizonyításában is látni fogjuk.

2.6. Megjegyzés. Fontos megjegyezni, hogy ez az $O(D)$ idő nem tartalmazza az összes útpár tényleges kiolvasását (tekintve, hogy akkor már az output is $O(n^2)$ méretű lenne), csak útjelzőket számolunk ki, ami alapján csúcsonként $O(n)$ időben kiolvashatók az útvonalak egy meghatározott módon. Így az összes útpár kiolvasása $O(n^2)$ időt vesz igénybe, de erre nem mindig van szükség: például a 3.2.1. szakaszban bemutatott algoritmusokban csak a csúcsok egy részéhez tartozó útpároknak is csak egy-egy kezdőszeletének a kiszámítása történik meg.

Az algoritmus szerkezete a következő.

2.7. Algoritmus (Suurballe)**.**

1. Minden pontba a legrövidebb irányított út keresése r-ből (Dijkstra), az így kapott fenyőt jelölje T;
2. *Új c' súlyfüggvény:* potenciál-eltolás a kapott értékekkel:
$$c'(uv) := c(uv) + d(r,u) - d(r,v)$$
(így a költség T élein 0, és mindenhol nemnegatív);
3. minden csúcshoz egyszerre: $p(v)$ és $q(v)$ útjelzők kiszámítása a 2.8. Algoritmussal ($O(D)$ időben).

Ezután tetszőleges v csúcsra az útpár kiolvasható az útjelzőkből $O(n)$ idő alatt (élenként $O(1)$) a 2.9. Algoritmussal. Így minden csúcsra az útpárok kiolvasása $O(n^2)$ időt vesz igénybe.

Most lássuk először részletesen a 3. fázist, majd a kiolvasást, utána néhány, értelmezést segítő vagy a bizonyításhoz szükséges megjegyzést, végül pedig a helyesség bizonyítását.

Az algoritmus során fenntartunk:

- $d(v)$ ideiglenes távolságcímkét minden csúcsra (mint a Dijkstra algoritmusban, és a cél az, hogy a végén minden v-re ez éppen a G_v-beli legrövidebb r-t út hossza legyen);
- a véglegesített csúcsok S halmazát;
- és T-nek a nem véglegesített csúcsokból álló részfáit.

2.8. Algoritmus ($p(v)$ és $q(v)$ útjelzők kiszámítása)**.**

1: ▷ *Inicializálás:*
2: $d(r) := 0$; $\forall v \in V \backslash \{r\}$-re: $d(v) := \infty$;
3: $S := \emptyset$;
4:
5: **while** $S \neq v \in V : d(v) < \infty$ **do**
6: ▷ *csúcs véglegesítése:*
7: legyen $v \in V \backslash \{r\}$ az a csúcs, amire $d(v)$ véges és minimális;
8: legyen T_v az a nem véglegesített részfa, amiben benne van a v (ebben v nem feltétlenül gyökér);
9: $S := S \cup \{v\}$;
10: T_v-t lecseréljük azokra a részfákra, amikre szétesik a v elhagyásával;
11: K:=$\{$olyan $uw \notin T$ élek: vagy $u = v$, vagy $u, w \in T_v$ és a szétvágás után u és w különböző részfába kerülnek$\}$; ▷ *most szétesők nemfaélek halmaza*
12: **for** $uw \in K$ **do** ▷ *a most szétesők nemfaélek feldolgozása:*
13: **if** $d(v) + c'(uw) < d(w)$ **then** ▷ *javítás, ha kell (vigyázat, nem $d(u)$-hoz, hanem $d(v)$-hez adunk $c'(uv)$-t!)*
14: $d(w) := d(v) + c'(uw)$;
15: $p(w) := u$;
16: $q(w) := v$;
17: **end if**
18: **end for**
19: **end while**

Az útjelzők kiszámításának futásideje: minden él legfeljebb egyszer kerülhet a K halmazba, tehát legfeljebb egyszer kell feldolgozni, tehát a javítás összesen $O(m)$ időben megtörténik; a legkisebb $d(v)$-vel rendelkező csúcs kiválasztása pedig megfelelő kupaccal implementálva összesen $O(n \log n)$ idő; tehát összesen $O(D)$ időt kapunk.

A következő algoritmus kiolvassa az útjelzők alapján az útpárokat az olyan t csúcsokra, melyekre $d(t) \neq \infty$.

Jelölje *szülő*(x) az x csúcs T-beli szülőjét.

2.9. Algoritmus (Az útpárok kiolvasása p, q értékekből egy t csúcsra)**.**

1: Kezdetben minden $x \in V$-re legyen jelölés$(x) := 0$; ▷ *ha több csúcs útpárjait is kiolvassuk, akkor elég egyszer beállítani (2.13. Következmény).*
2:
3: ▷ *útjelző q-k megjelölése (rendre $q(t)$, $q(q(t))$,..., r pontokat jelöljük meg[2]):*
4: $x := t$;
5: **while** $x \neq r$ **do**
6: jelölés(x):=1;
7: x:=q(x);
8: **end while**
9:
10: ▷ *t-től r-ig végiglépegetve kiolvassuk a két utat egymás után (utak jelölése: P_1, P_2):*
11: **for** $i = 1, 2$ **do** ▷ *a két útra*
12: $P_i := \emptyset$; $x := t$; ▷ *inicializálás*
13: **if** jelölés$(x) = 1$ **then** ▷ *ha x útjelző volt és még meg van jelölve, akkor $p(x)$ felé lépünk tovább:*
14: jelölés$(x) := 0$; ▷ *a most felhasznált útjelző jelölését töröljük*
15: $P_i := \{p(x)x\} \cup P_i$; ▷ *$p(x)x$ élet P_i elejére fűzzük*
16: $x := p(x)$; ▷ *továbblépünk $p(x)$ felé*
17: **else** ▷ *ha x nem útjelző, vagy már használtuk, akkor T-beli a szülője felé lépünk tovább:*
18: $P_i :=$ {szülő$(x)x$} $\cup P_i$; ▷ *szülő$(x)x$ élet P_i elejére fűzzük*
19: $x :=$szülő(x); ▷ *továbblépünk szülő(x) felé*
20: **end if**
21: **end for**

2.10. Megjegyzés. A 2.8. Algoritmusban az inicializálásból következik, hogy az első véglegesített pont mindig r lesz (ahogyan a 2.3. Algoritmusban is a G_t-ben a B' javítóút keresésére futtatott második Dijkstra futtatás is az r címkézésével (véglegesítésével) kezdődik).

2.11. Megjegyzés (Az útjelzők, a javítóutak, és a végleges útpár kapcsolata)**.** Figyeljük meg, hogy a 2.8. Algoritmusban a javítási lépés megfelel a G_w-ben egy w-be vezető jobb javítóút keresésének a $v \to u$ úton és az uw élen keresztül (mint azt majd a 2.16. Állítás bizonyításában is láthatjuk). Itt kihasználjuk, hogy a v-ből u-ba 0 költségű út vezet T_w-ben.

Az algoritmus végén egy x pontra: $p(x)x$ az az él, amin keresztül utoljára javítottunk; a $q(x)$ pedig az a csúcs, amin keresztül $p(x)$-be ér a $p(x)x$-et tartalmazó javítóút G_x-ben.

Másképp, a Suurballe algoritmus szemszögéből: $q(x)$ az a csúcs, aminek a véglegesítése okozta a $p(x)x$ él feldolgozását (ami a 2.16. Állítás szerint az optimális útpár egyik útjának

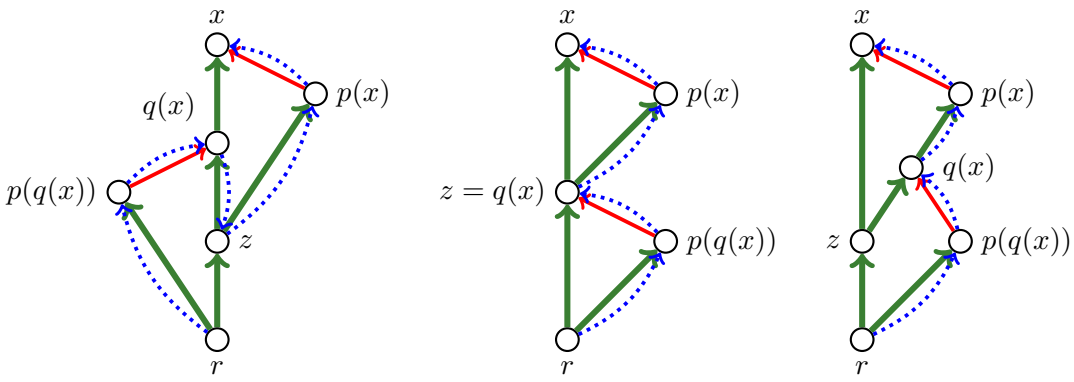
[2]Valóban eljutunk r-be $q(t)$, $q(q(t))$, ... lépegetéssel a 2.14. Megjegyzés miatt.

utolsó éle lesz); azaz az először véglegesített (tehát legkisebb d értékű) olyan pont, amit elhagyva T-ből x és $p(x)$ különböző részfába kerül; tehát a legkisebb d-értékű pont az x és $p(x)$ közötti egyértelmű T-beli, irányítatlan értelemben vett úton.

A kiolvasáskor a $q(x)$ vagy azt mutatja, hogy x-ből, vagy hogy $p(x)$-ből indulva a legrövidebb útpár útjain az r felé, hol kell letérni a T-beli útról az aktuális csúcs p-jének irányába.

Ezeknek az összefüggését így is láthatjuk: ha T-beli x-$p(x)$ úton nincs rajta az r csúcs, akkor a T-beli $P(T,x)$ út r-ből x-be, és a T-beli $P(T,p(x))$ út r-ből $p(x)$-be egy z pontig megegyeznek, és utána diszjunktak (ahol a z a T-ben az x-nek és a $p(x)$-nek az r-től legtávolabbi közös őse). Ezért a diszjunkt útpár útjait x-ből visszafelé követve, valahol le kell térnünk a $P(T,x)$, $P(T,p(x))$ utak valamelyikéről. A leheséges letérési pontok a T-beli x-$p(x)$ út pontjai, ezek közül pedig a minimális d értékűt fogjuk választani (mert $d(w) =$ a minimális javítóút költsége w-be).

A G_x-beli javítóút $q(x)$-nek a T-beli x-$p(x)$ úton belüli elhelyezkedésétől függően háromféleképpen viselkedhet, ezeket az eseteket a 2.2. ábrán láthatjuk. (Az ábra csak vázlat, az r-z szakasz és az r-$p(q(x))$ szakasz hasonló módon variálódhat.)



(a) $q(x)$ rajta van a T-beli z-x úton. Ekkor a G_x-beli javítóút biztosan tartalmaz $P(T,x)$-beli megfordított élet.

(b) $z = q(x)$. Ekkor a két út biztosan nem lesz csúcsdiszjunkt.

(c) $q(x)$ rajta van a T-beli z-$p(x)$ úton. Ekkor a $p(x)$ felé induló úton lesz másik $p(v)v$ típusú él, mielőtt újra eléri $P(T,x)$ utat.

2.2. ábra. A javítóút és a az útjelzők kapcsolata. Három eset lehetséges $q(x)$ elhelyezkedésétől függően a T-beli x-$p(x)$ úton. Jelölje z a T-ben az x-nek és a $p(x)$-nek az r-től legtávolabbi közös ősét. Az élek jelölése: zöld (vastag): T élei; piros (vékony): $p(x)x$ élek; kék (pöttyözött): javítóút G_x-ben.

2.12. Állítás. *A $q(x)$ útjelző mindig rajta van a legrövidebb éldiszjunkt r-x útpárból valamelyik úton.*

Bizonyítás: Indirekt tegyük fel, hogy a $q(x)$ nincs rajta egyik úton sem. A kiolvasás algoritmusa miatt ez csak úgy fordulhatna elő, ha lenne egy olyan $q^k(x) := q(q(\dots(q(x)))) \neq q(x)$,

ami rajta van vagy a T-beli $q(x)$-x úton, vagy a T-beli $q(x)$-$p(x)$ úton. Tehát a $q^k(x)$ rajta lenne a T-beli $p(x)$-x úton. Viszont a Suurballe algoritmus előbb véglegesíti $q^k(x)$-et, mint $q(x)$-et (mert a $q(x)$-nél előbb véglegesíti a $q(q(x))$-et, aminél előbb a $q(q(q(x)))$-et, stb.). Tehát a $q^k(x)$ véglegesítésétől külön részfába kerülne az x és a $p(x)$, azaz $q^k(x) = q(x)$, ami ellentmondás.

□

2.13. Következmény. *A 2.12. Állítás miatt: több csúcshoz tartozó útpárok kiolvasásakor a 2.9. Algoritmusban a jelölés(x) jelölőbitet nem kell minden csúcs útjainál külön inicializálni 0-ra, mert az útpár kiolvasásának a végére az 1-re állított biteket mind visszaállítjuk.*

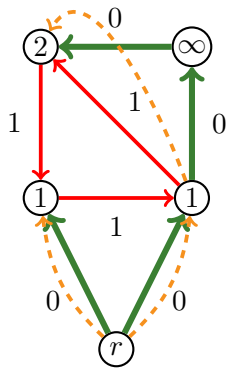
2.14. Állítás. *Tekintsük a következő élhalmazt:* $\{(q(x), x) : x \neq r, d(x) \neq \infty\}$, *ahol* $(q(x), x)$ *nem feltétlenül G-beli él. Ezek egy r gyökerű fenyőt határoznak meg.*

Bizonyítás: A $(q(x), x)$ virtuális élek által meghatározott $Q(V, E_q)$ gráfban minden $x \in V \backslash \{r\}$, $d(x) < \infty$ csúcsnak a befoka 1. Olyan y csúcsra, amire $d(y) = \infty$, nem illeszkedhet $(q(x), x)$ él, mert soha nem javítottuk a $d(y)$ értékét egy másik csúcson kereszül, tehát $q(y)$ nem kaphatott értéket; és így soha nem véglegesítettük, tehát $y = q(x)$ sem lehet. Tehát ezeket a csúcsokat elhagyhatjuk Q-ból.

Másrészt tudjuk, hogy Q aciklikus, mert $q(x)$-et előbb véglegesítette az algoritmus, mint x-et, tehát a véglegesítési sorrend a Q csúcsainak egy topologikus sorrendje.

Tehát a $(q(x), x)$ élek r gyökerű fenyőt alkotnak. □

Viszont a hasonlóan képzett $(p(x), x)$ élhalmaz már nem feltétlenül aciklikus, például a 2.3. ábrán látható gráfban sem az.



2.3. ábra. Egy gráf c' súlyfüggvénnyel, és a Suurballe-Tarjan eredményével. A csúcsokra írt számok a végső d értékek. Az élek jelölése: zöld (vastag): T élei; piros (vékony): $p(x)x$ típusú élek; narancssárga (szaggatott): $(q(x), x)$ virtuális élek. Az egyik csúcsba nem vezet irányított útpár. Figyeljük meg, hogy a piros élek irányított kört alkotnak.

2.15. Megjegyzés. Mivel minden r-t út költsége $d(r,t)$-vel csökken a potenciál-eltoláskor, ezért (a 2.5. Megjegyzéshez hasonlóan) az r-ből t-be futó legrövidebb útpár $d_2(t)$ összsúlya megadható

$$d_2(t) = d(t) + 2d(r,t)$$

alakban az algoritmus által kiszámolt értékekből, az útpárok tényleges kiolvasása nélkül.

A következő állítás a (már bizonyítottan optimális) $v \in V\backslash\{r\}$ csúcsonként G_v-ben javítóutat kereső 2.3. Algoritmus, és a 2.8. Suurballe algoritmus kapcsolatát mutatja. A bizonyításban minden v-re külön-külön "egymás mellett" futtatunk egy 2.3. javítóút kereső Dijkstra algoritmust G_v gráfon (c' súlyfüggvénnyel), és egy Suurballe algoritmust a G gráfon a c' súlyfüggvénnyel, és megmutatjuk, hogy a v útjainak szempontjából ugyanazokat a lépéseket végzik. (A bizonyítás szerkezete megegyezik a [9] cikkbelivel, csak kiegészítettem néhány gondolattal.)

2.16. Állítás (Suurballe és Tarjan [9]: Theorem 2.)**.** *A Suurballe algoritmus végén igazak a következők:*

1. $d(v) = d_v(r,v)$ *minden* v *csúcsra (ha nem létezik éldiszjunkt útpár* r*-ből* t*-be, akkor végtelen);*
2. *minden olyan* $v \in V\backslash\{r\}$*-re, amire* $d(v)$ *véges, létezik* G_v*-ben egy olyan* r*-*v *út, aminek a* (c' *szerinti) hossza* $d(v)$ *és az utolsó éle* $p(v)v$*.*

Bizonyítás: Legyen $v \in V\backslash\{r\}$ tetszőleges adott csúcs. Most belátjuk, hogy v-re igaz az állítás.

2.17. Definíció. A Suurballe algoritmus lefutásában nevezzünk (v szempontjából) **lényeges** csúcsnak egy l csúcsot, ha az l Suurballe-beli véglegesítésekor az l ugyanabban a nem véglegesített részfában volt, mint a v.

A lényeges csúcsok tehát mindaddig a v-vel közös nem véglegesített részfában maradnak, amíg nem véglegesítjük őket (tehát például a v után véglegesített csúcsok nem lényegesek).

Azt fogjuk belátni, hogy a G_v-ben futtatott Dijkstra algoritmus a lényeges csúcsokon ugyanolyan lépéseket végez, mint a Suurballe, azaz ugyanolyan sorrendben véglegesíti a lényeges csúcsokat, és ugyanolyan javítási lépéseket tesz a d, p értékeken (a nem lényegeseket pedig esetleg hamarabb véglegesíti, és nem feltétlenül egyeznek meg a megfelelő értékek).

Maga a v csúcs is lényeges, mert önmagával akkor és csak akkor van ugyanabban a nem véglegesített részfában, ha még nem véglegesítettük őt.

2.18. Definíció. Jelölje a bizonyítás során a G_v-n futtatott Dijkstra algoritmusban az ideiglenes távolság- illetve megelőző pont függvény aktuális értékét d_v és p_v, a *címkézett* (azaz véglegesített) pontok halmazát pedig S_{G_v}.

Most (a G gráfon futtatott) Suurballe csúcsvéglegesítési lépésenként haladva indukciószerűen megmutatjuk, hogy hogyan futtatható vele párhozamosan úgy a Dijkstra algoritmus a G_v gráfon, hogy igazak legyenek az alábbi Invariáns tulajdonságok.

2.19. Invariáns. *1. Minden x csúcsra: akkor és csak akkor $x \notin S_{G_v}$ (azaz címkézetlen a Dijkstra algoritmusban), ha x és v azonos részfában vannak a Suurballe-ban.*

2. Ha $x \notin S_{G_v}$, akkor $d_v(x) = d(x)$ és $p_v(x) = p(x)$ (azaz a Dijkstra és a Suurballe szerinti ideiglenes távolságcímkéjük és p-jük megegyezik), feltéve, hogy kapott már értéket valamelyik.

3. Ha $x \in S_{G_v}$, akkor pedig $d_v(x) = d(z)$, ahol z az a csúcs, aminek (a Suurballe-ban) a véglegesítése miatt került x és v különböző részfába; ha $x = z$, akkor $p_v(x) = p(z) = p(x)$.

Kezdetben igaz az invariáns (mert minden x csúcsra $x \notin S_{G_v}$ és $x \in T$ részfa, és $d(x)$, $p(x)$ még egyik algoritmusban sem kaptak (véges) értéket).

Indukciós feltétel: az invariáns teljesül egy y csúcs (a Suurballe-ban való) véglegesítéséig.

Most belátjuk, hogy a Dijkstra megfelelő futása mellett *az y csúcs Suurballe-beli véglegesítésekor* sem történik változás.

Két eset lehetséges (aszerint, hogy y-t címkézte-e már a Dijkstra):

1. eset: $y \in S_{G_v}$

   Az invariáns (indukciós feltétel) 1. része miatt y és v már *különböző részfában* voltak (azaz *y nem lényeges*).

   Tehát y véglegesítése nincs hatással a T_v-beli csúcsok címkéire (mivel nem választhat szét odavezető élet), így nem ronthatja el az invariánst.

   Az y csúcs Suurballe-beli véglegesítésével párhuzamosan tehát a G_v-n futó Dijkstra *nem tesz semmit.*

2. eset: $y \notin S_{G_v}$

   Az invariáns 1. része miatt y és v *azonos részfában* voltak eddig a pillanatig (azaz *y lényeges*); jelölje $F_{\text{régi}}$ ennek a részfának a csúcshalmazát. Legyen $F_{\text{új}}$ pedig annak a részfának a csúcshalmaza, amibe v kerül y véglegesítése után (ha $y = v$, akkor $F_{\text{új}}$ üres).

   Az y csúcs Suurballe-beli véglegesítésének megfelelő lépéssorozatként a G_v-n futó Dijkstra *először címkézi az y csúcsot, majd az $F_{régi} - F_{új}$ halmaz maradék csúcsait* (azaz a most keletkező új részfák közül az összes v-t nem tartalmazónak az összes csúcsát) a megfelelő sorrendben.

   Most két dolgot kell belátnunk: egyrészt, hogy a Dijkstra valóban címkézheti ezeket a csúcsokat; másrészt, hogy az invariáns fennmarad.

2.20. Állítás. *Az y a Dijkstraban minimális címkézetlen pont, azaz*

$$d_v(y) = \min\{d_v(x) : x \notin S_{G_v}\}.$$

Bizonyítás: A Dijkstra minden csúcsot vagy ugyanakkor, vagy előbb címkéz, mint a Suurballe, és most $y \notin S_{G_v}$.

Az invariáns 2. része szerint y véglegesítéséig minden $x \notin S_{G_v}$-re: $d(x) = d_v(x)$.

Tehát mivel a Suurballe a minimális d értékkel rendelkező y csúcsot választja ki véglegesítésre, így igaz az állítás. □

2.21. Állítás. *Minden $F_{régi} - F_{új}$ halmazbeli csúcs elérhető G_v-ben y-ból 0 súlyú úton.*

Bizonyítás: A G_v-ben a T-beli r-v utat megfordítottuk, tehát az így kapott T' egy v gyökerű fenyő. Tehát T'-ben az $F_{\text{régi}} - F_{\text{új}}$ összes csúcsa az y leszármazottja lesz, amik tehát elérhetők T' élein 0 súlyú úton. □

A Suurballe a következő éleket dolgozza fel az y véglegesítésekor:

- y-ból induló nemfaéleket;
- az $F_{\text{régi}} - F_{\text{új}}$ és az $F_{\text{új}}$ közötti éleket;
- $F_{\text{régi}} - F_{\text{új}}$ új részfái között futó éleket.

Minden fenti uv él feldolgozásakor ez történik: ha $d(v) < d(y) + c'(uv)$, akkor $d(v) := d(y) + c'(uv)$.

Ezzel párhuzamosan a Dijkstra a következő lépéseket végzi.

- Az y minimalitása (2.20. Állítás) miatt: először címkézheti az y csúcsot, és javít az $\overrightarrow{yv}$ éleken: ha $d_v(v) < d_v(y) + c'(uv)$, akkor $d_v(v) := d_v(y) + c'(uv)$. Speciálisan a T' élein $c' = 0$, tehát az y-nak minden T'-beli g gyerekére $d_v(g) := d_v(y)$.
- Ismét az y minimalitása miatt: most az előbb $d_v(y)$ értéket kapott csúcsokat címkézheti sorban, és minden ilyen g csúcsra javít a kimenő $\overrightarrow{gv}$ éleken: ha $d_v(v) < d_v(y) + c'(gv)$, akkor $d_v(v) := d_v(y) + c'(gv)$. Tehát a g-nek a T'-beli gyerekei is $d_v(y)$ értéket kapnak.
- ...És így tovább, minden olyan csúcsot sorban megcímkézhet, ami elérhező y-ból 0 súlyú úton, tehát a 2.21. Állítás szerint minden $F_{\text{régi}} - F_{\text{új}}$ halmazbeli u csúcsot megcímkézhet, és mindegyikből kimenő $\overrightarrow{uv}$ éleken javít: ha $d_v(v) < d_v(y) + c'(uv)$, akkor $d_v(v) := d_v(y) + c'(uv)$.

Így az invariáns továbbra is fennáll, mert pontosan azokat a csúcsokat címkézi a Dijkstra, amik a Suurballe-ban a v-től különböző címkézetlen részfába kerültek (és

az y-t), és ezek a csúcsok az invariáns 3. része szerinti d_v, p_v értékeket kapják; a még címkézetlen csúcsokat pedig ugyanazon élek mentén, ugyanazokra az értékekre javítja a Dijkstra, mint a Suurballe, tehát ezekre a csúcsokra pedig igaz lesz az invariáns 2. része.

Így beláttuk, hogy az invariáns végig fennmarad.

Az algoritmus végén az összes csúcs címkézett lesz a Dijkstraban, tehát mindegyikre az Invariáns 3. része fog teljesülni. Tehát a lényeges csúcsokra a d, p értékek megegyeznek a két algoritmus szerint (mivel ilyenkor teljesül, hogy $x = z$), a nem lényegesekre pedig nem feltétlenül.

Speciálisan a v is lényeges, tehát az algoritmus végén $d(v) = d_v(v)$ és $p(v) = p_v(v)$. A G_v-n futtatott Dijkstra algoritmusról már láttuk, hogy optimális megoldáshoz tartozó $d_v(v)$, $p_v(v)$ értéket ad. Tehát teljesül az állítás.

□

2.22. Megjegyzés. A v csúcs útjain a q típusú útjelzők a v szerinti lényeges csúcsok közül kerülnek ki.

2.23. Tétel. *Suurballe algoritmusa minden csúcsba optimális útpárt talál.*

Bizonyítás: Mivel a 2.4. Állítás szerint a javítóutas a 2.3. Algoritmus optimális megoldást ad, ezért a 2.16. Állatásból már 2.11. Megjegyzés miatt következik, hogy a q útjelzők segítségével kiolvashatók az optimális útpárok, tehát Suurballe algoritmusa optimális megoldást talál. □

2.3. Tarjan adatszerkezete

A 2.8. Algoritmus futásidejének kiszámítása során eltekintettünk attól a fontos részlettől, hogy hogyan választjuk ki a K halmazba kerülő feldolgozandó éleket. Minden élet megvizsgálni $O(m)$ idő lenne, ami már elrontaná a futásidőt, így Tarjan és Suurballe olyan adatszerkezetekkel implementálták az algoritmust, amivel a futásidő a fent kiszámított marad, ahogy azt látni fogjuk ennek a szakasznak a végén.

Amit tárolnunk kell:

- *Mélységi és befejezési számok.* Futtassunk az inicializálási szakaszban egy mélységi bejárást a T fán, és minden x csúcsra jegyezzük meg a mélységi (azaz elérési) és befejezési számokat: $M(x)$, $B(x)$. (Ennek ideje: $O(n)$, de elég egyszer futtatni az algoritmus elején. Tárhely igény: $O(n)$.)
  Ez azért hasznos, mert $O(1)$ időben eldönthető, hogy egy x csúcs leszármazottja-e v csúcsnak: akkor leszármazottja, ha $M(x) > M(v)$ és $B(x) < B(v)$; és a leszármazottak intervallumot alkotnak M szerint.

- *Nem véglegesített részfák.* Minden x csúcsra ezeket tároljuk:
  - *szülő*(x) (ha x gyökér a részfában, akkor *nil*)
  - *gyerekek*(x): a gyerekek oda-vissza ciklikusan láncolt listája (így egy gyerek törlése $O(1)$ időben lehetséges).

  Inicializálás: az egész T előállítása ebben a formátumban ($O(n)$ idő és tár).
  Karbantartás v törlésekor:
  - v törlése a *gyerekek*(*szülő*(x)) listából;
  - v gyerekeinek a szülő-pointerének átállítása *nil*-re.

  (Ez összesen az algoritmus során $O(n)$ időt vesz igénybe).

- *Éllista.* Az oda-vissza láncolt éllista minden v csúcsra a még nem feldolgozott, nem T-beli, v-re illeszkedő (ki- és bemenő) élek. Minden csúcs listája az élek másik végpontjának $M(x)$ értéke szerinti *monoton növekvő sorrendben* tartalmazza az éleket. Inicializálás: radix rendezés ($O(m)$ időben, mivel $\{M(x) : x \in V\} = [1..n]$).[3]

Az adatszerkezetek karbantartása és a feldolgozandó élek megtalálása egy v csúcs véglegesítésekor:

1. A v-re illeszkedő éleket töröljük, ezek közül a kimenőket feldolgozzuk. (Összesen az algoritmusban: $O(m)$ idő.)

2. Az új nem véglegesített részfáknál:

   - a v *szülőjéhez* tartozó részfának minden x csúcsára megnézzük az x-re illeszkedő éleket: amelyik szomszéd leszármazottja v-nek, azt az élet feldolgozzuk és töröljük az éllistából. Amelyik nem leszármazottja v-nek, azzal *nem csinálunk semmit*; az ilyen nem feldolgozandó "rossz" élek vizsgálatát nevezzük **elvesztegetett lépés**nek. Mivel a v leszármazottai intervallumot alkotnak, az első nem-leszármazottnál befejezhetjük a csúcsra illeszkedő élek vizsgálatát.
   - v-nek minden y *gyerekére*: az y-hoz tartozó részfának minden x csúcsára: hasonlóan. Itt x éllistájában a nem feldolgozandó élek alkotnak intervallumot (amiknek a másik csúcsa is az aktuális részfában van, azaz y leszármazottai), tehát így járjuk be: elkezdjük az élek vizsgálatát lista elejéről, és addig dolgozzuk fel és töröljük az éleket, amíg y leszármazottjába nem ütközünk; ekkor elkezdjük a lista végéről, és ott is addig haladunk tovább, amíg "rossz" élhez nem érünk.

[3] Vagy mélységi bejárás közben, mikor $M(x)$ sorrendjében vesszük sorra a pontokat, mindegyik x pontnál felvehetjük az x-re illeszkedő éleket a másik végpont éllistájának a végére; ez is $O(m)$-ben helyettesíti a rendezést.

2.24. Definíció. Nevezzük a részfák feldolgozása során **Lépés**nek a következőt: addig dolgozzuk fel sorban az éleket az aktuálisan vizsgált csúcs éllistájában, amíg vagy egy elvesztegetett lépést teszünk, vagy a csúcs listájának a végére érünk.

A részfák feldolgozását *"párhuzamosan"* végezzük, sorban minden fában *egy-egy Lépést* teszünk.

Így az utolsónak maradt (legnagyobb) fát nem kell befejezni (mert az onnan egy másik fába átlépő éleket már megtaláltuk a másik végüknél). A váltogatás miatt viszont oda kell figyelni, hogy mikor törlünk egy uw élet mindkét végpont listájából, akkor az egyik lista éleinek a vizsgálatában nem éppen ott tartottunk-e, mert akkor az elem törlésekor gondoskodnunk kell a rá mutató pointerről is: át kell helyezni a törlendő elem előtti vagy utáni pontra (attól függően, hogy előre vagy hátra haladunk éppen az adott listában). Egy él törlése egy éllistából ezzel együtt is $O(1)$ idő.

2.3.1. Lépésszám elemzése a részfák átvizsgálásánál

Az egyetlen, aminek még nem állapítottuk meg az idejét: egy csúcs véglegesítésekor keletkezett új részfák csúcsainál az éllisták átvizsgálása. A részfák "párhuzamos" feldolgozásán múlik, hogy igazán hatékony becslést kapunk erre.

Egy Lépés ideje: $O(1) + O(1) \cdot$ (feldolgozott élek száma). Tehát a részfák vizsgálatának *teljes ideje*: $O(m)$+Lépések száma. Így a következőkben a Lépéseket fogjuk megszámolni az egész algoritmusban.

Nézzük meg, mi történik egy v csúcs véglegesítésekor.

Jelölje S a v-t tartalmazó részfát a v véglegesítése előtt, és $S_0, S_1, \ldots S_k$ azokat a részfákat, amikre S szétesik a v véglegesítésével, mégpedig úgy, hogy a $szülő(v) \in S_0$, és $|S_1| \geq |S_2| \cdots \geq |S_k|$. (Az S_0 részfában minden éllista feldolgozása eltér a többi fabelitől, ezért érdemes külön kezelni.)

A párhuzamos feldolgozás miatt a legnagyobb fát nem kell befejeznünk, tehát a legnagyobb fában ugyanannyi lépést végzünk, mint a második legnagyobb fában, tehát minden fában legfeljebb a lépések felét végezzük. Ezért ha az egyik (nem feltétlenül a legnagyobb) fában tett Lépések megszámolásától eltekintünk, akkor legfeljebb a felére csökken a megszámolt Lépések száma a valódi Lépésszámhoz képest. Tehát egy, a későbbiekben megválasztott fát ki fogunk hagyni a Lépések számolásából, és a végén az így kapott Lépésszámot megszorozzuk 2-vel. Ezen kívül a csúcsvéglegesítéseket kell még megszámolni, ez összesen az algoritmusban legfeljebb $L_{\text{csúcs}} := n$ darab.

A gondolat mindenhol az lesz, hogy úgy határozzuk meg azt, hogy melyik fától tekintünk el, hogy ezáltal egy felső becslést kapunk az $\frac{|S_i|}{|S|}$ értékre, és ebből az S_i-vel azonos típusú fákban az algoritmus során tett összes Lépés számára.

Tekintsük először az $S_2, \ldots, S_k$ fákat. Ezekben minden x csúcs éllistáján legfeljebb

2 Lépés alatt végigmegyünk (egy előrefelé az első elvesztegetett lépésig, és egy visszafelé hasonlóan).

Tudjuk, hogy ($i = 2 \dots k$-ra:) $|S_i| \leq \frac{|S|}{2}$, mert ha az egyik fa nagyobb az eredeti felénél, akkor az lenne az S_1, tehát nem szerepelne ezek között. Tehát egy adott x csúcsot figyelve az algoritmus során, amikor ilyen típusú fában Lépéseket végzünk rajta, akkor a fa mérete legalább a felére csökkent (és 1-nél nem lehet kisebb a csúcsszám). Tehát összesen legfeljebb $2 \log n$ Lépést végezhetünk egy adott x csúcsnál, azokat összeszámolva, mikor ilyen típusú fákba kerül.

Tehát összesen az egész algoritmusban legfeljebb $2n \log n =: L_2$ darab Lépést végzünk az $S_2, \dots, S_k$ típusú fák csúcsainál.

Most tekintsük S_0 és S_1 fákat.

Jelölje

$$\alpha := \max\left(2, \frac{m}{n \log(1 + \frac{m}{n})}\right)$$

értéket.

Két eset lehetséges S_0 méretét tekintve:

1. S_0 *kicsi*: $\alpha|S_0| \leq |S|$. Ekkor S_1 részfától tekintünk el, tehát elég az S_0 lépéseit számolni.

   Minden $x \in S_0$ csúcsnál legfeljebb $m(x)+1$ Lépést végzünk (ahol $m(x)$ az x-re illeszkedő élek száma). Ez az eset azt jelenti, hogy egy $x \in T_0$ csúcsot tartalmazó részfa mérete v véglegesítésének hatására legalább az α-adrészére csökkent, tehát (a fentihez hasonlóan) minden adott x csúcshoz az algoritmus során az ilyen részfákban legfeljebb $(m(x)+1) \log_\alpha n$ Lépés tartozik, azaz összesen minden csúcsra $(2m + n) \log_\alpha n =: L_0$ darab Lépés (mivel $\sum_{x \in V} m(x) = 2m$).

2. S_0 *nagy*: $\alpha|S_0| \geq |S|$. Ekkor S_0-tól tekintünk el, tehát most az S_1 lépéseit számoljuk.

   Minden $x \in S_1$-nél legfeljebb 2 Lépés történik. Egy adott x-et tartalmazó részfa mérete ilyenkor legalább $\alpha/(\alpha - 1)$-adrészére csökken, tehát összesen minden csúcsra az algoritmus során legfeljebb $2n \log_{\alpha/(\alpha-1)} n =: L_1$ darab Lépést végzünk ilyen típusú fában.

Így az S_0 és S_1 fákban tett Lépések számára is kaptunk becslést.

Tehát összesen az eddigiekből a következő lépésszámot kapjuk:

$$L_{\text{csúcs}} + 2(L_2 + L_0 + L_1) = n + 4n \log n + (4m + 2n) \log_\alpha n + 4n \log_{\alpha/(\alpha-1)} n$$

ahol az utolsó tag

$$O\left(\frac{n \log n}{\log[\alpha/(\alpha - 1)]}\right) = O\left(\frac{n \log n}{\log[1 + 1/\alpha]}\right) = O(\alpha n \log n)$$

és

$$\log \alpha = \max(1, \log(m/n) - \log\log(1 + m/n)) = \Omega(\log(1 + m/n))$$

tehát az összeg:

$$O(n \log n + m \log_{(1+m/n)} n) = O(m \log_{(1+m/n)} n) = O(D).$$

Tehát valóban az ígért futásidőt kaptuk.

2.4. Az algoritmus általános esetben

Mivel a 3.2. fejezetben irányítatlan gráfra, pontdiszjunkt útpárok keresésére szeretnénk használni az algoritmust, most ismerkedjünk meg az ilyen irányú általánosítási lehetőségekkel, és néhány továbbival.

2.4.1. Pontdiszjunkt eset

Pontdiszjunkt útpárok keresése irányított gráfban egy jól ismert trükk segítségével visszavezethető az éldiszjunkt esetre: minden G-beli v csúcsot "széthúzunk" egy v_{ki} és egy v_{be} csúcsra, amik között egy 0 költségű $\overrightarrow{v_{be}v_{ki}}$ él fut, és a v csúcsba befelé mutató élek a v_{be} csúcsra, a kifelé mutatók pedig a v_{ki} csúcsra fognak illeszkedni az átalakított G_e gráfban. (Az r gyökérnél ez a módosítás elhagyható.)

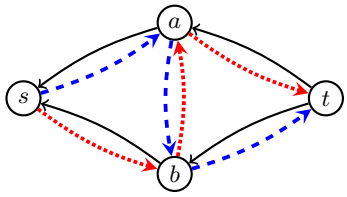
Így minden, a v csúcson áthaladó G-beli út G_e-beli megfelelője használni fogja a $\overrightarrow{v_{be}v_{ki}}$ élet, tehát két ilyen nem lehet éldiszjunkt, azaz bármely G_e-beli éldiszjunkt megoldás G-ben pontdiszjunkt lesz, és visszafelé: minden G-beli pontdiszjunkt megoldásnak megfelel egy G_e-beli éldiszjunkt megoldás.

Így csak n darab új csúcs és n darab új él keletkezik, és az új gráf $O(m)$ időben előállítható, tehát az algoritmus futásidejére és a szükséges tárra tett becslésünk nem módosul.

2.4.2. Irányítatlan eset

Az irányítatlan esetet is visszavezethetjük egy egyszerű konstrukcióval: minden irányítatlan uv élet cseréljünk két ellentétes irányítású élre ($\overrightarrow{uv}$, $\overrightarrow{vu}$).

Ez ugyan okozhat problémákat, mert egy éldiszjunkt útpár egy adott élet akár mindkét irányba is használhat egyszerre, például a 2.4. ábrán látható gráfban nem lesz éldiszjunkt az $r \to a \to b \to t$ és az $r \to b \to a \to t$ út megfelelője a visszaalakított gráfban, de ez minden útpárnál kijavítható (vegyük az utak szimmetrikus differenciáját). A vizsgált hálózatokban ez nem okozott gondot, mert a legrövidebb út keresése miatt ez az eset nem is fordulhat elő pozitív élköltség mellett (ami ésszerű elvárás lehet egy valós hálózattal szemben), és az általunk főként vizsgált pontdiszjunkt esetben sem lehet ilyen. Így m új él keletkezik, tehát a futásidő ebben az esetben sem változik.



2.4. ábra. Nem éldiszjunkt utak az oda-vissza megirányított gráfban.

2.4.3. Multigráfok

Irányított multigráfokra közvetlenül átalakítható az algoritmus, ahol tehát megengedünk többszörös, akár mindkét irányba irányított éleket (aminek speciális esete az irányítatlan gráf visszavezetésekor kapott oda-vissza megirányított gráf). Elég $p(v)$ fogalmát kicsit máshogyan definiálni: most nem a v-t megelőző csúcs lesz az úton, hanem a v-be vezető utolsó él.

2.4.4. Negatív élek

Egy további általánosítási lehetőség, hogy a c súlyfüggvényben megengedünk negatív értékeket is.

Ha a súlyfüggvény konzervatív, akkor elég a Dijkstra első futása helyett a Bellman-Ford algoritmust alkalmazni (ami el is dönti, hogy c konzervatív-e), mivel a második futásakor már a nemnegatív c'súlyfüggvénnyel dolgozunk. Ez némi lassulást eredményezhet, mivel a legrövidebb utak keresését a Bellman-Ford algoritmus $O(nm)$ időben végzi.

Ha viszont c nem konzervatív, akkor a feladat értelemszerű átfogalmazása NP-teljes, ahogy azt a következőkben láthatjuk. [9]

3. Probléma (Legrövidebb sétapár létezése probléma (LSLP))**.** Tegyük fel, hogy létezik $G(V, E)$ irányított gráfban negatív összköltségű irányított kör. Döntsük el, hogy adott s, t pontokhoz létezik-e legkisebb összsúlyú két éldiszjunkt s-t séta (vagy tetszőlegesen kis súlyút találhatunk).

2.25. Tétel. *A legrövidebb sétapár létezése probléma NP-teljes.*

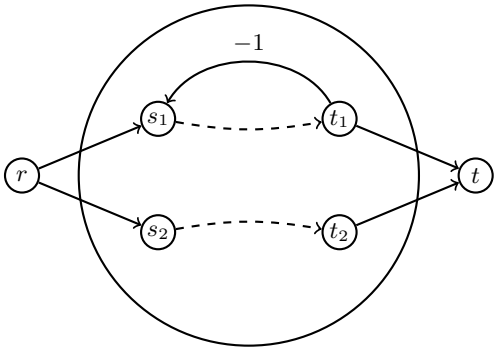
Bizonyítás: A feladat NP-beli, mivel egy k-nál kisebb költségű út megfelelő tanú.

Az NP-nehézség bizonyításához visszavezetünk rá egy (többtermékes folyamok elméletéből) ismert NP-teljes feladatot:

4. Probléma (Két-út feladat)**.** Adott $G(V, E)$ irányított gráf, és $s_1, s_2, t_1, t_2 \in V$ különböző csúcsok. Léteznek-e olyan s_1-t_1 és s_2-t_2 utak, melyek csúcsdiszjunktak?

Tegyük fel, hogy adott a két-út feladatnak egy példánya G gráfban, ebből előállíthatunk egy G' gráfban egy LSLP példányt a következőképpen. Először a csúcsdiszjunkt-éldiszjunkt

visszavezetés érdekében alkalmazzuk a fentebb említett csúcs-széthúzást, így minden csúcsból egy v_{be} és egy v_{ki} példány keletkezik, kivéve s_1, s_2, t_1, t_2 csúcsokból, amiket nem szükséges kettébontani. Majd vegyünk fel két új pontot: s-et és t-t, és 5 új élet: ss_1, ss_2, t_1t, t_2t, és t_1s_1. Legyen c élsúly minden élen 0, kivéve az új t_1s_1 élt, ahol -1.



2.5. ábra. A két-út probléma visszavezetése a legrövidebb sétapár létezése problémára. $c(t_1s_1) = -1$, minden más él súlya 0.

2.26. Állítás. *G-ben akkor és csak akkor létezik a kívánt csúcsdiszjunkt útpár, ha G'-ben nem létezik legrövidebb sétapár, azaz tetszőlegesen kis súlyú sétapár található s-ből t-be.*

Bizonyítás: Feltehetjük, hogy G-ben létezik út s_1-ből t_1-be, azaz G'-ben létezik negatív kör.

Tegyük fel, hogy G-ben létezik két diszjunkt út, nevezzük az s_1-t_1 utat P_1-nek, a másikat P_2-nek. Ekkor G'-ben s-ből t-be akármilyen kis súlyú sétát alkothatunk úgy, hogy a P_1 út és a t_1s_1 él által alkotott kört elég sokszor hozzávesszük az (ss_1, P_1, t_1t) úthoz. Fordítva: csökkenő súlyú séták sorozata csak az előbb említett módon jöhet létre, mert nincs másik negatív él a gráfban, tehát ha létezik tetszőlegesen kis súlyú éldiszjunkt sétapár, akkor azok egy része G-ben pontdiszjunkt útpárt határoz meg. □

□

3. fejezet

Algoritmusok kis költségű független fapár keresésére

Ebben a fejezetben olyan algoritmusokkal ismerkedünk meg, amik minél kisebb költségű független fapárt keresnek. Először egy optimális megoldást adó módszert tekintünk, ami viszont reménytelenül lassan fut; majd két gyors heurisztikát, friss kutatási eredményekből; majd összehasonlításképpen egy régebbi, jóval lassabb, de általában jobb eredményt adó heurisztikát.

3.1. Optimális megoldást adó ILP

A feladat modellezhető egészértékű lineáris programozási feladattal az alábbiakban vázolt módon. [7]

Tekintsük azt a $D(V, \overrightarrow{E})$ irányított gráfot, amit úgy kapunk, hogy G-ben minden $uv \in E$ él helyett egy $\overrightarrow{uv}$ és egy $\overrightarrow{uv}$ irányított élet veszünk, amiknek a súlya $c(uv)$. Ebben tehát olyan r felé irányított T_1, T_2 fenyőket keresünk, amelyek függetlenek.

A megoldásban egy x_e^{s,T_i} útváltozó azt mutatja, hogy az s-ből r-be futó T_i-beli út tartalmazza-e az e élet; $y_e^{T_i}$ faváltozó pedig azt, hogy a T_i fában szerepel-e az e él.

Az egyenletek jelentése:

- A 3.1. azt biztosítja, hogy minden változó indikátorváltozó legyen.
- Utakra vonatkozó egyenletek:
  - a 3.2. egyenlet a folyam-tulajdonságot garantálja (ami a fa-tulajdonsággal együtt út-tulajdonságot eredményez);
  - a 3.3. az útpárok éldiszjunktságát (azaz hogy egy élen az s-ből induló utak közül legfeljebb az egyik fabeli haladhat át az egyik irányba).
- Fákra vonatkozó egyenletek:

  - 3.4. azt jelenti, hogy e él csak akkor lehet az útban, ha a fában is benne van;
  - 3.5. pedig a fa-tulajdonságot (minden v csúcsnak egy szülője van, azaz egy v-ből kimenő él szerepel a fában).

- Pontfüggetlen esetben feltesszük a 3.6. feltételeket is, amik az útpárok pontdiszjunktságát garantálják (legfeljebb az egyik s-ből induló fabeli út tartalmazhat v-ből kimenő élet).

- Végül a 3.7. célfüggvény éppen $c(T_1, T_2)$-t minimalizálja.

Egészségi feltétel:
$\forall v \in V, \forall s \in V \backslash \{r\}, \forall T_i \in \{T_1, T_2\}, e \in \overrightarrow{E}$-re:

$$x_e^{s,T_i}, y_e^{T_i} \in \{0,1\}. \tag{3.1}$$

Utakra vonatkozó egyenletek:
Folyam-tulajdonság:
$\forall s \in V \backslash \{r\}$ startcsúcsra, $T_i \in \{T_1, T_2\}$-re, $\forall v \in V$ (köztes) csúcsra:

$$\sum_{\overrightarrow{vw} \in \overrightarrow{E}} x_{\overrightarrow{vw}}^{s,T_i} - \sum_{\overrightarrow{wv} \in \overrightarrow{E}} x_{\overrightarrow{wv}}^{s,T_i} = \begin{cases} 1 & \text{ha } v = s \\ -1 & \text{ha } v = r \\ 0 & \text{különben} \end{cases} \tag{3.2}$$

Éldiszjunktság:
$\forall \overrightarrow{vw} \in E$ élre, $\forall s \in V \backslash \{r\}$ startcsúcsra:

$$x_{\overrightarrow{vw}}^{s,T_1} + x_{\overrightarrow{vw}}^{s,T_2} + x_{\overrightarrow{wv}}^{s,T_1} + x_{\overrightarrow{wv}}^{s,T_2} \leq 1 \tag{3.3}$$

Fákra vonatkozó egyenletek:
Kapcsolat az útváltozók és a faváltozók között:
$\forall e \in \overrightarrow{E}$ élre, $\forall s \in V \backslash \{r\}$ startcsúcsra, $\forall T_i \in \{T_1, T_2\}$-re:

$$x_e^{s,T_i} \leq y_e^{T_i} \tag{3.4}$$

Fa-tulajdonság:
$\forall v \in V$ csúcsra, $\forall T_i \in \{T_1, T_2\}$-re:

$$\sum_{\overrightarrow{vw} \in \overrightarrow{E}} y_{\overrightarrow{vw}}^{T_i} \leq \begin{cases} 1 & \text{ha } v \neq r \\ 0 & \text{ha } v = r \end{cases} \tag{3.5}$$

Pontfüggetlen esetben ezt is feltesszük az utakra:
$\forall s \in V \backslash \{r\}$ startcsúcsra, $\forall v \in V \backslash \{r, s\}$ köztes csúcsra:

$$\sum_{\overrightarrow{vw} \in \overrightarrow{E}} (x_{\overrightarrow{vw}}^{s,T_1} + x_{\overrightarrow{vw}}^{s,T_2}) \leq 1 \tag{3.6}$$

Célfüggvény:

$$\min \sum_{e \in \vec{E}} [c(e) \cdot \sum_{s \in V \setminus \{r\}} (x_e^{s,T_1} + x_e^{s,T_2})]. \tag{3.7}$$

Ez megoldható (exponenciális időben), így lassú, de optimális r-független feszítőfapárt találó algoritmust kaptunk.

3.2. Heurisztikák

Ahogyan az 1.2.1. fejezetben láttuk, ha egy 2-összefüggő gráfnak adott egy nyílt fülfelbontása, akkor az 1.22. Tétel bizonytásában bemutatott módszerekkel s-t rendezéssé/számozássá, s-t irányítássá, majd független fapárrá alakíthatjuk élszámban lineáris időben. Valójában az s-t irányítás tényleges előállítására nincs szükség, mert az adott fülfelbontás sorrendjében felépítve a gráfot, az s-t rendezés szerint felépíthetők a fák a következőképpen.

3.1. Algoritmus (Független fapár építése fülfelbontásból)**.**

- Az egyetlen élből álló triviális füleket nem tekintjük.
- $P_0 = (s, v_1, v_2, \ldots, v_{q-1})$ alapkörből vegyünk az sv_{q-1} él kivételével minden élt a T_1 fához, és az sv_1 él kivételével minden élt a T_2 fához.
- A soronkövetkező $P_j = (u_0, u_1, \ldots, u_m)$ fülből ismét a két szélső él kivételével minden él bekerül mindkét fába, a szélsők pedig a rendezéstől függően: ha $u_0 \prec u_m$, akkor u_1u_0 kerül T_1-be, és $u_{m-1}u_m$ T_2-be; különben fordítva.

Mivel valójában mindig csak az előző fülek rendezésére van szükség, így a fülfelbontásból már a rendezéssel egyszerre építhetjük a fákat is.

A kérdés már csak az, hogy az algoritmus alapjául szolgáló fülfelbontást hogyan keressük meg. A most következő heurisztikák erre adnak különböző megoldásokat.

3.2. Megjegyzés. Fülfelbontás és s-t rendezés keresésére Tarjan 1972-ben élszámban lineáris idejű algoritmust adott [11] mélységi keresés segítségével, ami azon alapul, hogy minden v csúcshoz megkeresi a *lowpoint*(v) csúcsot, ami a v-nek a legkisebb mélységi számmal rendelkező szomszédja. Azóta többen is adtak lineáris idejű algoritmust a feladatra, például 2013-ban Schmidt [8] bemutatott egy egyszerű, út-központú ("path-based") mohó algoritmust (ami szintén mélységi fát használ, amit gyökér felé irányít, és a gyökértől elfelé megirányított nemfaélekből kiindulva mohón utakat keresve állít elő egy fülfelbontást).

Tehát fülfelbontás és s-t rendezés keresése, ebből következően független fapár keresése is lehetséges lineáris időben, de ezek az algoritmusok nem érzékenyek a súlyozásra. Az ebben a fejezetben található heurisztikák ugyan lassabbak, de az optimalizálási feladatra keresnek minél jobb megoldást (még mindig polinomiális futásidővel).

Mindegyik ismertetett heurisztika egyfajta mohó megközelítést alkalmaz, és fülről fülre fokozatosan építi fel a fülfelbontást. A 3.1. Algoritmus alapján a heurisztikákban a már kész részgráfhoz a fülfelbontással párhuzamosan folyamatosan építhetjük az s-t rendezést és a független fapárt is.

Először két gyors ($O(m \log_{(1+m/n)} n)$ és $O(n^2)$ időben futó) heurisztikával ismerkedünk meg, ami a Suurballe-Tarjan algoritmust is használja; majd egy n^2-szer lassabbal, ami ugyan általában jobb megoldást ad, de nagy gráfokra reménytelenül lassú.

3.2.1. Heurisztikák a legrövidebb útpárok alapján

Ebben az alszakaszban olyan heurisztikákat láthatunk (a 2015-ös a [10] cikkből), amik a minden $v \neq r$ csúcsból r-be futó legkisebb összsúlyú diszjunkt útpárokat használják fel, tehát mindegyik a Suurballe-Tarjan algoritmus futtatásával kezdődik, ami $O(D)$ idejű (ahol $D = m \log_{(1+m/n)} n$ a Dijkstra algoritmus futásideje d-edfokú kupaccal). Tehát nem kaphatunk $O(D)$-nél jobb becslést az algoritmus összidejére, de ezt nem is vártuk, hiszen a hagyományos routing algoritmusokban használt legrövidebb utak fájának keresése is ilyen nagyságrendű időbe telik Dijkstra algoritmussal.[1]

Először bemutatunk egy gyors heurisztikát, ami $O(D)$ időben fut, majd ennek továbbfejlesztését, ami jobb fapárt ad, de egy kicsit lassabb: $O(n^2)$ idejű.

A minimális összsúlyú éldiszjunkt r-t útpár súlyát továbbra is jelölje $d_2(t)$.

Minimális d_2 értékű csúcs útjait választva

A soronkövetkező fület úgy építjük fel, hogy kiválasztjuk azt a v csúcsot a rész-fülfelbontás által még fedetlenek közül, aminek a $d_2(v)$ értéke a legkisebb (ha ilyenből több is van, akkor azt vesszük, amit a Suurballe algoritmus először véglegesített), és vesszük az r-be futó minimális diszjunkt útpár útjainak a v-től az első már fedett pontig tartó szegmensét. Ezen útszegmensek uniója legyen az új fül.

3.3. Megjegyzés. Az útpárok d_2 összsúlyát minden t csúcsra kiszámolhatjuk a 2.15. Megjegyzés szerint $d_2(t) = d(t) + 2d(r,t)$ alakban, ahol a $d(t)$ és $d(r,t)$ értékeket megkapjuk a Suurballe algoritmus futása során, az útpárok tényleges kiolvasása nélkül. Tehát tetszőleges v csúcsra $d_2(v)$ kiolvasható $O(1)$ időben.

Ugyan a Suurballe a d értékek sorrendjében véglegesíti a csúcsokat (azaz elég lehet a d ismerete a véglegesítési sorrend megállapításához is), de ezek esetleg megegyezhetnek, ha c' értéke valahol 0. Ez apró változtatásokkal kezelhető, de az egyszerűség kedvéért most a Suurballe véglegesítési sorrendjével dolgozunk.

[1] Persze a Dijkstra algoritmus Fibonacci-kupaccal még gyorsabban, $O(m + n \log n)$ időben fut.

Mivel az útpárok útjainak csak egy-egy kezdőszeletére van szükség, ezért azok kiolvasásakor nem kell a teljes 2.9. Algoritmust végrehajtani. (Ezen múlik a futásidő, mert különben mind az $O(n)$ darab fül keresésekor $O(n)$ időbe telik az utak kiolvasása, ami csak $O(n^2)$ futásidő-becslést eredményez.)

3.4. Állítás. *Ha minden fülhöz a minimális $d_2(v)$ értékű fedetlen v csúcsot (és egyenlőség esetén ezek közül a Suurballe által legkorábban véglegesítettet) választjuk, akkor a v útjain egyetlen olyan q típusú útjelző sem lesz, ami még fedetlen.*

Bizonyítás: Mivel G irányítatlan, és a 2.12. Állítás szerint $q(v)$ rajta van a v legrövidebb útpárján, tehát $d_2(q(v)) \leq d_2(v)$ (mert a v útpárja egyben a $q(v)$ útpárja is).

Ha $d_2(q(v)) < d_2(v)$, akkor a fülfelbontás építésében a v kiválasztásakor $q(v)$-nek már fedettnek kell lennie, különben a v helyett $q(v)$-t választanánk, tehát ez esetben valóban fedett $q(v)$ (és hasonlóan $q(q(v)), q(q(q(v))), \ldots$ is).

Így csak a $d_2(q(v)) = d_2(v)$ esetet kell megvizsgálnunk. Ekkor az algoritmusunk választása szerint v a Suurballe által először véglegesített csúcs a minimális d_2 értékű fedetlenek közül. De a Suurballe algoritmus előbb véglegesítette $q(v)$-t, mint v-t, tehát $q(v)$ fedett, mert különben őt választotta volna az algoritmus v helyett. □

Tehát a 2.9. Algoritmusban *nem* kell a q-k bejelölését végrehajtani, és az útpár élein elég addig lépegetni, amíg fedetlen csúcshoz nem érünk (ami élenként $O(1)$ idő).

Az algoritmus pszeudokóddal az alábbi 3.5. Algoritmusként látható.

3.5. Algoritmus (Legkisebb $d_2(v)$ értékű csúcs útjait használó heurisztika)**.**

1: Suurballe(G); ▷ *p útjelzők kiszámítása a legrövidebb útpárokhoz, a végső és a Dijkstra által talált távolságcímkék és a véglegesítési sorrend tárolása*
2: **for** $v \in V \backslash \{r\}$ **do**
3: $d_2(v)$ kiszámítása (a Suurballe során eltárolt értékekből a 3.3. Megjegyzés alapján);
4: **end for**
5:
6: ▷ *Fülek keresése:*
7: $V^* := V - \{r\}$ sorbarendezése d_2, és azon belül a Suurballe véglegesítési sorrendje szerint; ▷ *fedetlen csúcsok halmazának inicializálása*
8: $G_f := \{r\}$; ▷ *fülfelbontás inicializálása*
9: $i := 0$; ▷ *fülek indexe*
10: **while** $V^* \neq \emptyset$ **do**
11: ▷ *Új fül:*
12: $v := V^*$ első eleme; ▷ *minimális d_2 értékű (és azon belül az először véglegesített) fedetlen csúcs választása*
13: $P'_1(v) := P(T_1, v)$ kiolvasva az első $(V - V^*)$-beli elemig, ami $=: x$;
14: $P'_2(v) := P(T_2, v)$ kiolvasva az első $(V - V^*)$-beli elemig, ami $=: y$;

15: $P_i := P_1'(v) \cup P_2'(v) =: (x, v_1, \dots, v, \dots, v_{k_i}, y)$;

16:

17: ▷ *jelöljük át P_i fül végpontjait $a \prec b$-re (és a 2. és az utolsó előtti csúcsot $v_{start} \prec v_{end}$-re):*

18: **if** $x \prec y$ **then**

19: $a := x$; $b := y$;

20: $v_{\text{start}} := v_1$; $v_{\text{end}} := v_{k_i}$;

21: **else** ▷ *ha $y \prec x$:*

22: $a := y$; $b := x$;

23: $v_{\text{start}} := v_{k_i}$; $v_{\text{end}} := v_1$;

24: **end if**

25:

26: ▷ *Rendezés építése:*

27: a' := az a csúcs rákövetkezője $\prec$ szerint;

28: $a \prec v_{\text{start}} \prec \cdots \prec v \prec \cdots \prec v_{\text{end}} \prec a'$;

29:

30: ▷ *Fák építése:*

31: $T_1 := T_1 \cup (a, \dots, v, \dots, v_{\text{end}})$; ▷ *$b$ nincs benne*

32: $T_2 := T_2 \cup (v_{\text{start}} \dots, v, \dots, b)$; ▷ *$a$ nincs benne*

33:

34: $V^* := V^* - \{u \in P_i\}$;

35: $G_f := G_f \cup P_i$;

36: $i := i + 1$;

37: **end while**

Az első $P_0 = (r, v_1, \dots, v_{k_0}, r)$ fülnél (ami egy kör) a rendezés nem következik még az előzőkből, ezért ott legyen $r \prec v_1 \prec \cdots \prec v_{k_0}$. Minden más ugyanúgy működik, mint a későbbi füleknél.

Futásidő: a csúcsok sorbarendezése $O(n \log n) < O(D)$ idő; a fülek kiolvasása és adminisztrációja élenként $O(1)$ idő, tehát összesen $O(n)$ (mert a végső fülfelbontásnak $< 2n$ éle van). Tehát a futásidő $O(D)$ a Suurballe-Tarjan miatt.

Javítás részben rendezéssel

3.6. Észrevétel. Az 1.23. Lemma bizonyításában és az előző 3.5. Algoritmusban fülfelbontásból s-t rendezést képeztünk, de egy-egy fül valójában csak részbenrendezést ad meg közvetlenül a csúcsok egy részhalmazán (ami valóban mindig kiegészíthető teljes rendezéssé).

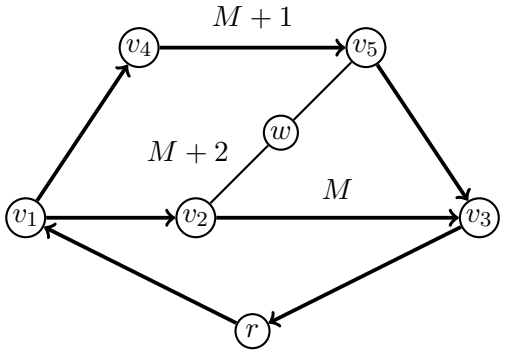
Ez alapján átalakíthatjuk a 3.5. Algoritmust úgy, hogy s-t rendezés helyett csak s-t részbenrendezést tartson számon a csúcsokon.

3.7. Megjegyzés. Részbenrendezés helyett praktikusabb lehet az 1.20. Állítás szerint vele ekvivalens s-t irányítással dolgozni (azaz $x \prec y \iff$ létezik $x \to y$ itányított út). Az alábbiakban vegyesen használom a két fogalmat: a magyarázatoknál és az implementációban az irányítás a kézenfekvőbb, az algoritmus leírásánál pedig a rendezés (az előző algoritmussal való összehasonlítás végett).

Irányítással megfogalmazva tehát a részbenrendezést így tartjuk karban: "legyen $x \prec y$" értékadáskor behúzunk egy $\overrightarrow{xy}$ élet; "igaz-e $x \prec y$" lekérdezéskor pedig megvizsgáljuk, hogy vezet-e irányított út x-ből y-ba.

Ha az algoritmust átalakítjuk úgy, hogy csak a fülfelbontásból következő részbenrendezést építse fel, akkor új fül hozzávételénél előfordulhat, hogy a fül két végpontja nincs rendezve a részbenrendezésben. Ekkor bármelyik irányban rendezve a fület részbenrendezést kapunk, tehát eldönthetjük, hogy melyiket választjuk.

Ez azért javíthat jelentősen a fapárok súlyán, mert a két lehetőségből adódó fapárokban az utak hossza lényegesen eltérhet, például tekintsük a 3.1. ábrát, ahol a 3.5. Algoritmusban alkalmazott teljes rendezés éppen a rosszabb megoldást választaná.



3.1. ábra. Példa arra, hogy részbenrendezéssel jobb eredményt kapunk. Súlyozás: M nagy, a jelöletlen élek 0 költségűek. Utoljára a w ponthoz tartozó fület vesszük be a fülfelbontásba, ebben a pillanatban a rendezés iránya az ábrán látható irányításnak felel meg a rendezést és a részbenrendezést használó algoritmusban is. Részbenendezés esetén most a $v_2 - w - v_5$ fület v_5 felé irányítjuk meg, így w fabeli útjainak az összköltsége $M+2$ lesz; míg az eredeti rendezéses módszernél, mivel $v_1 - v_2 - v_3$ fület hamarabb választjuk, mint $v_1 - v_4 - v_5 - v_3$ fület, ezért $v_5 \prec v_2$, tehát $v_5 - w - v_2$ fület v_2 irányába rendezzük, és így w útjainak összsúlya $3M+3$ lesz (minden más csúcsnál marad ugyanaz).

Nem rendezett végpontok esetén tehát kiszámítjuk a fül végpontjaiból a gyökérbe vezető út hosszát a piros és a kék fában, és a következő értékeket (a 3.5. Algoritmus jelöléseivel): $c(P(T_1, x)) + c(P(T_2, y))$ és $c(P(T_2, x)) + c(P(T_1, y))$, és ezek közül a kisebbikhez tartozó részbenrendezést választjuk.

3.8. Megjegyzés. Egy x csúcs $c(P(T_i, x))$ értkeit többféleképpen megkaphatjuk:

- vagy $P(T_i, x)$ kiolvasásával, mikor szükségünk van az adott $c(P(T_i, x))$ értékre: ez $O(n)$ idő lekérdezésenként, azaz az algoritmus során összesen $O(n^2)$ idő és $O(1)$ tár (mivel $O(n)$ darab fülkeresés történik, és egy fülnél legfeljebb 2 darab csúcsra kell lekérdeznünk a $c(P(T_i, x))$ értéket);
- vagy az algoritmus fenntarthat minden x csúcsra $c_{T_1}(x) = c(P(T_1, x))$, $c_{T_2}(x) = c(P(T_2, x))$ értékeket, ezek egy újonnan fedett csúcs esetén $O(1)$ időben meghatározhatók mindkét fára a fabeli szülő megfelelő értékéből, azaz összesen az algoritmus során $O(n)$ idő és $O(n)$ tár.

Egyik sem módosít sem a futásidőre, sem a tárra tett becslésünkön, így a pszeudokódban az utak kiolvasását használó módszert láthatjuk.

3.9. Megjegyzés. Ugyan az 1.2.1. szakaszban a létezés bizonyítása csak rendezéssel működik, de mivel minden részbenrendezés kiegészíthető teljes rendezéssé, amiből mindig képezhető fapár, és a fák folytatásához csak a végpontok sorrendjére van szükség, ezért a megszokott módon építhetjük itt is a fákat az algoritmus közben.

Az algoritmus módosított változata tehát a következő.

3.10. Algoritmus (Legkisebb $d_2(v)$ heurisztika részbenrendezéssel)**.**

1: Suurballe(G); ▷ *p útjelzők kiszámítása a legrövidebb útpárokhoz, a végső és a Dijkstra által talált távolságcímkék és a véglegesítési sorrend tárolása*
2: **for** $v \in V \backslash \{r\}$ **do**
3: $d_2(v)$ kiszámítása (a Suurballe során eltárolt értékekből a 3.3. Megjegyzés alapján);
4: **end for**
5:
6: ▷ *Fülek keresése:*
7: $V^* := V - \{r\}$ sorbarendezése d_2, és azon belül a Suurballe véglegesítési sorrendje szerint; ▷ *fülfelbontás inicializálása*
8: $i := 0$; ▷ *fülek indexe*
9: **while** $V^* \neq \emptyset$ **do**
10: ▷ *Új fül:*
11: $v := V^*$ első eleme; ▷ *minimális d_2 értékű (és azon belül aminimális d értékű) fedetlen csúcs választása*
12: $P_1'(v) := P(T_1, v)$ kiolvasva az első $(V - V^*)$-beli elemig, ami $=: x$;
13: $P_2'(v) := P(T_2, v)$ kiolvasva az első $(V - V^*)$-beli elemig, ami $=: y$;
14: $P_i := P_1'(v) \cup P_2'(v) =: (x, \dots v \dots, y)$;
15:

▷ *jelöljük át P_i fül végpontjait $a \prec b$-re (és a 2. és az utolsó előtti csúcsot $v_{start} \prec v_{end}$-re):*
if $x \prec y$ **then**
  $a := x$; $b := y$;
  $v_{\text{start}} := v_1$; $v_{\text{end}} := v_{k_i}$;
else
  if $y \prec x$: **then**
    $a := y$; $b := x$;
    $v_{\text{start}} := v_{k_i}$; $v_{\text{end}} := v_1$;
  else ▷ *ha x és y nem állnak relációban egymással*
    $c_1 := c(P(T_1, x)) + c(P(T_2, y))$; ▷ *az úthosszak kiszámítása $O(n)$ idő*
    $c_2 := c(P(T_2, x)) + c(P(T_1, y))$;
    if $c_1 < c_2$ **then**
      $a := x$; $b := y$;
      $v_{\text{start}} := v_1$; $v_{\text{end}} := v_{k_i}$;
    else
      $a := y$; $b := x$;
      $v_{\text{start}} := v_{k_i}$; $v_{\text{end}} := v_1$;
    end if
  end if
end if

▷ *Részbenrendezés építése:*
$a \prec \cdots \prec v \prec \cdots \prec b$; ▷ *itt csak a b-hez képest rendezzük őket*

▷ *Fák építése:*
$T_1 := T_1 \cup (a, \ldots, v, \ldots, v_{\text{end}})$; ▷ *b nincs benne*
$T_2 := T_2 \cup (v_{\text{start}}, \ldots, v, \ldots, b)$; ▷ *a nincs benne*

$V^* := V^* - \{u \in P_i\}$;
$G_f := G_f \cup P_i$;
$i := i + 1$;
end while

Futásidő: A kritikus rész az, hogy mind az $O(n)$ darab fülválasztás után az irányítás meghatározásához meg kell vizsgálnunk, hogy az x, y végpontok között van-e valamelyik irányba irányított út a már felépített fülfelbontásban. Mivel egy olyan fülfelbontás, amiben nem szerepelnek 1 hosszú fülek, az ritka (ha n' csúcsa van, akkor $< 2n'$ éle van), ezért $O(n)$ időben eldönthető, hogy van-e két pont között irányított út (pl. mélységi kereséssel). Tehát

a futásidő $O(n^2)$.

3.2.2. BR algoritmus

Egy másik megközelítés, amit Balasubramanian és Ramasubramanian 2006-os algoritmusa használ [1] (ami az előbb bemutatott heurisztikákat is inspirálta), hogy a következő fület úgy keressük minden lépésben, hogy a fül súlya, és a már megépített részfákon a fülnek a gyökérig való meghosszabbítása együtt legyen minimális. Ez ugyan általában valamivel jobb megoldást ad, mint a 3.2.1. szakaszbeli algoritmusok, de sokkal lassabb: $O(n^2(D))$ idejű, így nagy gráfokra használhatatlan. (Ez a heurisztika nem használja a Suurballe algoritmust, csak a Dijkstra-t.)

A soronkövetkező fül választása a következő módon történik. Minden lehetséges x, y már fedett csúcspárra (amiből van kimenő fedetlen él) meghatározzuk az őket összekötő minimális fedetlen utat. Az új fül legyen ezek közül az, amire x és y rendezésétől függően $c_{T_1}(x) + c(\text{legrövidebb } x\text{-}y \text{ fül}) + c_{T_2}(y)$, vagy $c_{T_2}(x) + c(\text{legrövidebb } x\text{-}y \text{ fül}) + c_{T_1}(y)$ minimális (ahol $c_{T_i}(x) = c(P(T_i, x))$).

Mivel szükségünk lesz $c_{T_i}(x)$ értékekre, így a 3.10 Algoritmusnál a 3.8. Megjegyzéshez hasonlóan itt is dönthetünk:

- vagy minden $c_{T_i}(x)$ lekérdezéskor kiolvassuk $P(T_i, x)$ utat és összeadjuk az élsúlyokat: ez $O(n)$ idő lekérdezésenként, azaz az algoritmus során összesen $O(n^3)$ idő és $O(1)$ tár (mivel $O(n)$ fülkeresés történik, és minden fülkereséskor $O(n)$ darab csúcsra kell lekérdeznünk a $c_{T_i}(x)$ értéket);

- vagy az algoritmus fenntarthat (az s-t rendezésen és a T_1, T_2 kezdő-részfákon kívül) minden csúcsra $c_{T_1}(x)$, $c_{T_2}(x)$ értékeket is. Ezek egy újonnan fedett csúcs esetén $O(1)$ időben meghatározhatók mindkét fára a fabeli szülő megfelelő értékéből, azaz az ilyen típusú értékek lekérdezése az algoritmus során összesen $O(n)$ időbe és $O(n)$ tárba kerül.

Tehát ennél az algoritmusnál már általában érdemes az értékek számontartását választani (bár a futásidő becslésén egyik sem változtat), ezért a pszeudokódban is feltüntettem ezen értékek frissítését.

3.11. Algoritmus (BR algoritmus)**.**

1: $V^* := V$; ▷ *fedetlen csúcsok halmazának inicializálása*
2: $G_f := \{r\}$; ▷ *fülfelbontás inicializálása*
3: $c_{T_1}(r) := 0$; $c_{T_2}(r) := 0$;
4:
5: ▷ *Az első fül:*
6: $P_0 := (r, v_1, \ldots, v_{k_0}, r) :=$ a legrövidebb, r-et tartalmazó kör;

(jelölés: $v_0 := r$; $v_{k_0+1} := r$;)
▷ *Rendezés építése:*
$r \prec v_1 \prec \cdots \prec v_{k_0}$;
▷ *Fák építése:*
$T_1 := T_1 \cup (r, v_1, \ldots, v_{k_0})$;
$T_2 := T_2 \cup (v_1, \ldots, v_{k_0}, r)$;
▷ *A fül csúcsaira $c_{T_1}(v_j)$, $c_{T_2}(v_j)$ értékek kiszámítása:*
for $j = 1 \ldots k_i$ **do**
    $c_{T_1}(v_j) := c_{T_1}(v_{j-1}) + c(v_{j-1}v_j)$;
end for
for $j = k_i \ldots 1$ **do**
    $c_{T_2}(v_j) := c_{T_2}(v_{j+1}) + c(v_{j+1}v_j)$;
end for
$V^* := V^* - \{u \in P_0\}$;
$G_f := G_f \cup P_0$;

▷ *Fülek keresése:*
$i := 1$; ▷ *fülek indexe*
while $V^* \neq \emptyset$ **do**
    ▷ *Új fül keresése:*
    for all $a, b \in V(G_f)$ **do** ▷ *fedett csúcsokra*
        $d(a,b) := \min\{c(P) : P$ út, $P - \{a,b\} \subseteq V^*\}$;
        $P(a,b) := \arg\min\{c(P) : P$ út, $P - \{a,b\} \subseteq V^*\}$;
    end for
    $x, y := \arg\min_{a \prec b}\{c_{T_1}(a) + d(a,b) + c_{T_2}(b)\}$; ▷ *emiatt $x \prec y$*
    $P_i =: (x, v_1, \ldots, v_{k_i}, y) := P(x,y)$;

    ▷ *most $x \prec y$*
    ▷ *Rendezés építése:*
    $x' :=$ az x rákövetkezője $\prec$ szerint;
    $x \prec v_1 \prec \cdots \prec v_{k_i} \prec x'$;
    ▷ *Fák építése:*
    $T_1 := T_1 \cup (x, v_1, \ldots, v_{k_i})$;
    $T_2 := T_2 \cup (v_1, \ldots, v_{k_i}, y)$;
    ▷ *A fül csúcsaira $c_{T_1}(v_j)$, $c_{T_2}(v_j)$ értékek kiszámítása:*
    for $j = 1 \ldots k_i$ **do**
        $c_{T_1}(v_j) := c_{T_1}(v_{j-1}) + c(v_{j-1}v_j)$;
    end for

45: **for** $j = k_i \ldots 1$ **do**
46: $c_{T_2}(v_j) := c_{T_2}(v_{j+1}) + c(v_{j+1}v_j)$;
47: **end for**
48: $V^* := V^* - \{u \in P_i\}$;
49: $G_f := G_f \cup P_i$;
50: $i := i + 1$;
51: **end while**

Futásidő: a lényeges lépés minden fül választásánál a legrövidebb utak kiszámítása, például $O(n)$ darab Dijkstra algoritmussal. A fülkeresések száma $O(n)$, így kapjuk az $O(n^2D)$ futásidőt.

3.12. Megjegyzés. Az algoritmus elején a legrövidebb r-et tartalmazó kör számítását például a következőképpen végezhetjük.

Az r-re illeszkedő minden rw élre végezzük el a következőt: keressünk a $G - \{rw\}$ gráfban legrövidebb r-w utat, és ehhez az úthoz vegyük hozzá rw élet.

Az így kapott körök közül a legkisebb összsúlyút válasszuk.

Ez összesen $O(D \cdot N(r))$, ahol $N(r) = O(n)$ a gyökér szomszádainak a száma, azaz $O(D \cdot n) < O(n^2D)$. Tehát ez nem változtat a futásidő becslésén.

3.13. Megjegyzés. Természetesen ez az algoritus is továbbfejleszthető részbenrendezés használatával a 3.2. szakaszbelivel analóg módon.

3.2.3. A heurisztikák összehasonlítása két példán

A heurisztikák értékeléséhez bevezetjük a következő fogalmat:

$$Gap'(T_1, T_2) := \frac{c(T_1, T_2)}{\sum_{v \in V \setminus \{r\}} d_2(v)}$$

ahol T_1, T_2 az adott heurisztika által talált fapár lesz.

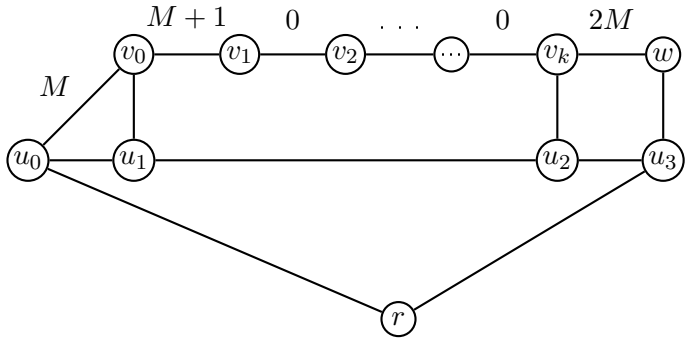
Ezen kívül vizsgálni fogjuk a $MaxGap(T_1, T_2)$ értéket is a példákban.

A BR és a minimális d_2 választó heurisztikák összehasonlítására most bemutatunk két példát: az egyikben az egyik, a másikban a másik teljesít egyértelműen jobban, amiből az is következik, hogy egyik sem ad mindig optimális fapárt.

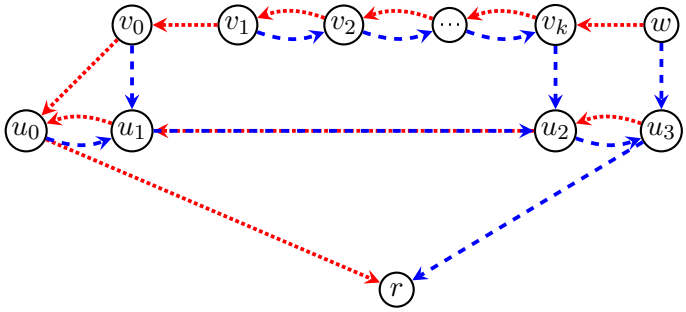
Mérések alapján [10] általában az a jellemző, hogy ugyan a BR teljesít jobban, de a részbenrendezéses módszer eredménye már nem marad le jelentősen a BR-éhez képest, és random generált síkgráfokon és kisvilág-tulajdonságú gráfokon (400 csúcsig vizsgálva) pedig szinte megegyező eredményt ad. Futásidőben viszont a d_2 heurisztikák sokkal jobbak.

Egy példa, ahol a d_2 heurisztika jobb fapárt ad, mint a BR algoritmus

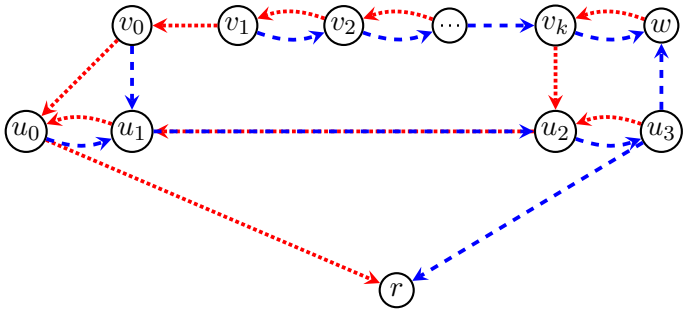
A 3.2. ábrán látható gráfot úgy súlyoztuk, hogy három él kivételével minden élsúly 0: $c(u_0v_0) = M$, $c(v_0v_1) = M + 1$, és $c(v_{k-1}v_k) = 2M$ (ahol M konstans, k-t pedig növeljük).



(a) Súlyozás (a jelöletlen élek súlya 0).



(b) Minimális d_2 heurisztikával kapott fapár



(c) BR algoritmussal kapott fapár

3.2. ábra. Példa, ahol BR rosszabbul teljesít: BR-nél $\lim_{k\to\infty} Gap' = 4$, d_2-nél $\lim_{k\to\infty} Gap' = 2$.

Tehát a legrövidebb útpárok összege a gráfban

$$\sum_{v\in V\setminus\{r\}} d_2(v) = M + k(M+1) + 2M = k(M+1) + 3M.$$

A d_2 heurisztikával kapott fákra (részbenrendezés használata esetén is ugyanez):

$$c(T_1, T_2) = M + k(2M+1) + 4M + 1 = k(2M+1) + 5M + 1$$

tehát

$$\lim_{k\to\infty} Gap'(T_1, T_2) = \lim_{k\to\infty} \frac{k(2M+1) + 5M + 1}{k(M+1) + 3M} = 2.$$

A BR algoritmus esetén pedig:

$$c(T_1, T_2) = M + 2M + (k-1)(4M+1) = k(4M+1) - M - 1$$

tehát

$$\lim_{k\to\infty} Gap'(T_1,T_2) = \lim_{k\to\infty} \frac{k(4M+1)-M-1}{k(M+1)+3M} = 4$$

azaz itt a BR heurisztika rosszabbul teljesít.

A legrosszabbul járó csúcs szemszögéből (a $MaxGap(T_1,T_2)$ célfüggvénnyel) hasonló a helyzet:
a d_2 heurisztikánál

$$MaxGap(T_1,T_2) = Gap_{T_1,T_2}(w) = \frac{4M+1}{2M} \approx 2$$

a BR algoritmusnál pedig

$$MaxGap(T_1,T_2) = Gap_{T_1,T_2}(v_1) = \frac{4M+1}{M+1} \approx 4$$

tehát eszerint is 2-szer jobb a d_2 heurisztika.

Egy példa, ahol a BR algoritmus teljesít jobban, mint a d_2 heurisztika

Érdekes, hogy a minimális d_2 értékű csúcshoz tartozó fület hozzávevő heurisztikánál a mérések során soha nem haladta meg a 1,3-at a Gap' érték, sőt a részbenrendezést használó változatnál a 1,15-öt sem, pedig elméletben tetszőlegesen nagy lehet (és hasonlóan a $MaxGap$ érték is, pedig a mérések során az sem haladta meg az 1,5-öt, sőt a részbenrendezést használó változatnál a 1,19-et sem.)

Például tekintsük a 3.3. ábrát, ahol az 1.3.1 szakaszban már látott "Lépcső" gráfot súlyozzuk, de most úgy, hogy $v_{i-1}v_i$ él költsége $M+i\varepsilon$ ($\forall i = 1\ldots k$-ra). Itt a d_2 heurisztikánál

$$Gap'(T_1,T_2) = \frac{M\frac{k(k+1)}{2} + \varepsilon\cdot O(k^3)}{Mk + \varepsilon\cdot O(k^2)} > \frac{k}{2}$$

tehát ha M tetszőleges konstans, k értékét növeljük, és $\varepsilon < \frac{1}{k^3}$, akkor

$$\lim_{k\to\infty} Gap'(T_1,T_2) = \infty.$$

Ugyanezen a gráfcsaládon a BR algoritmus esetén páratlan k-ra minden csúcs útja legfeljebb ε-nal lehet hosszabb a legrövidebb útpárnál, tehát ha M tetszőleges konstans, k értékét növeljük a páratlan számokon, és $\varepsilon < \frac{1}{k^3}$, akkor

$$\lim_{k\to\infty} Gap'(T_1,T_2) < \lim_{k\to\infty} \frac{Mk + \varepsilon\cdot O(k^2) + \varepsilon k}{Mk + \varepsilon\cdot O(k^2)} = 1$$

ráadásul páratlan k-ra optimális fapárt talál.

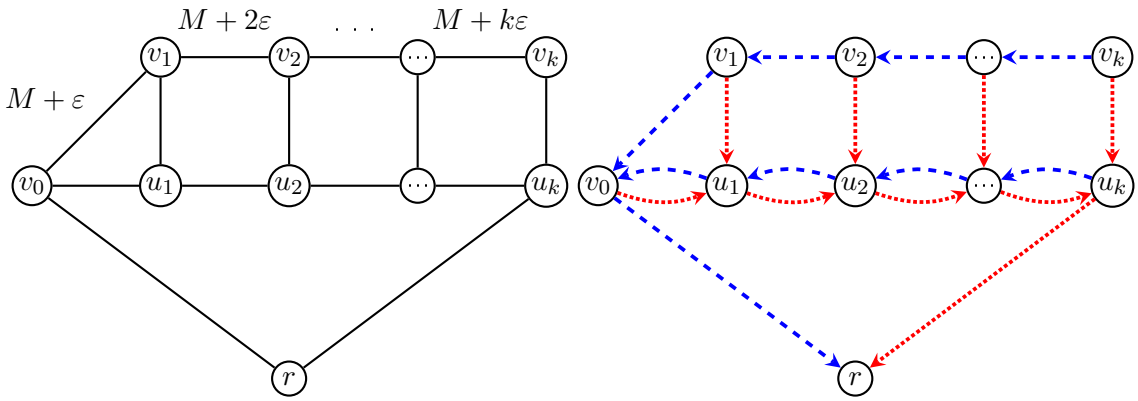
A $MaxGap(T_1,T_2)$ célfüggvény is hasonló eredményt mutat:
a d_2 heurisztikánál

$$\lim_{k\to\infty} MaxGap(T_1,T_2) = \lim_{k\to\infty} Gap_{T_1,T_2}(v_k) = \lim_{k\to\infty} \frac{kM + \varepsilon k(k+1)/2}{M+\varepsilon k} = \infty$$

a BR algoritmusnál pedig

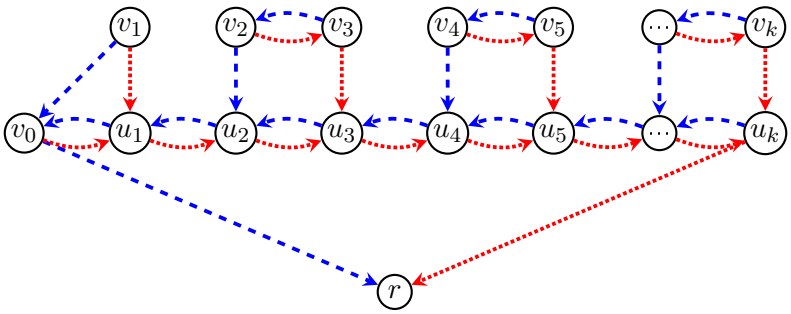
$$\lim_{k\to\infty} MaxGap(T_1,T_2) = \lim_{k\to\infty} Gap_{T_1,T_2}(v_2) = \lim_{k\to\infty} \frac{M+2\varepsilon}{M+3\varepsilon} = 1$$

tehát itt is ugyanazt kaptuk.



(a) "Lépcső" gráf $M + i\varepsilon$ súlyozással (a jelöletlen élek súlya 0).

(b) Minimális d_2 heurisztikával kapott fapár



(c) BR algoritmussal kapott fapár (páratlan k-ra)

3.3. ábra. A minimális d_2 heurisztikával tetszőlegesen nagy lehet $Gap'(T_1, T_2) > k/2$ (ha M konstans, k nagy, $\varepsilon < \frac{1}{k^2}$ kicsi).

Irodalomjegyzék

[1] R. Balasubramanian, S. Ramasubramanian: *Minimizing averagepath cost in colored trees for disjoint multipath routing*, IEEE ICCCN (2006), 185–190.

[2] Fehér Dániel: *Független fák keresése hálózatokban*, BSc szakdolgozat, ELTE (2014).

[3] A. Itai, R. Rodeh: *The multi-tree approach to reliability in distributed networks*, Proc. 25th Annu. IEEE Symp. on Foundations of Computer Sciences (1984), 137–147.

[4] A. Itai, M. Rodeh: *The multi-tree approach to reliability in distributed networks*, Inf. and Comput., vol. 79 (1988), no. 1, pp. 43–59.

[5] Király Zoltán: *Algoritmuselmélet* jegyzet (2014).
http://www.cs.elte.hu/~kiraly/Algelm.pdf

[6] Nagy Tamás: *Hálózati folyamok* jegyzet.
http://www.inf.unideb.hu/kmitt/konvkmitt/algoritmusok/book.xml.html

[7] S. Ramasubramanian, H. Krishnamoorthy, M. Krunz: *Disjoint multipath routing using colored trees*, Computer Networks, vol. 51, no. 8, pp. 2163–2180, (2007).

[8] J. M. Schmidt: *A Simple Test on 2-Vertex- and 2-Edge-Connectivity*, Information Processing Letters 113 (7) (2013a): 241–244.

[9] J. W. Suurballe, R. E. Tarjan: *A Quick Method for Findig Shortest Pairs of Disjoint Paths*, Networks, vol. 14 (1984), 325–336.

[10] Tapolcai J., Rétvári G., Babarczi P., Bérczi-Kovács E., Kristóf P., Enyedi G.: *Scalable and Efficient Multipath Routing: Complexity and Algorithms*, 23rd IEEE International Conference on Network Protocols (ICNP) (2015), pp. 1–10.

[11] R. E. Tarjan: *Depth-first search and linear graph algorithms*, SIAM Journal on Computing, 1(2) (1972), 146–160.

(Az internetes hivatkozással megadott források a 2016. 05. 20.-i állapotukban értendők.)