DIPLOMAMUNKA

Hatékony pivot algoritmusok hálózati folyamfeladatokra

Molnár-Szipai Richárd

Témavezető: **Dr. Illés Tibor**
egyetemi docens
BME Matematika Intézet,
Differenciálegyenletek Tanszék

BME
2012

Tartalomjegyzék

Köszönetnyilvánítás

Szeretném megköszönni témavezetőmnek, Illés Tibornak, hogy felhívta a figyelmemet erre az érdekes témára, és hogy észrevételeivel és tanácsaival segített a szakdolgozat elkészítésében.

Továbbá köszönöm a családomnak, a BME oktatóinak, munkatársainak és mindenkinek, aki hozzájárult tanulmányaim sikeres elvégzéséhez.

1. fejezet

Bevezető

1.1. A modell alkalmazási területei

A hálózati folyam feladat az elmúlt 70 év egy felettébb népszerű kutatási területe, rendkívül sok publikációnak adott témát. Ez főleg annak köszönhető, hogy egy intuitív, a gyakorlatban széles körben alkalmazható, egyszerű matematikai modellről van szó.

A hálózati folyam természetes modellje olyan problémáknak, ahol egy rendszerben anyag áramlik bizonyos keletkezési helyektől bizonyos felhasználási helyek felé. Az előbbi helyeket forrásoknak, az utóbbiakat nyelőknek nevezzük. Anyag alatt sokféle dolgot érthetünk: lehet szó csővezetékben áramló folyadékról, utcákon mozgó járművekről, vagy számítógépes hálózaton közvetített információról. A hálózat egyes csomópontjai között az anyag szállítására alkalmas médiumok vannak (cső, utca, kábel). Ezek a legegyszerűbb modellben egyirányú áramlást engednek meg, és rendelkeznek egy kapacitással, amit az időegység alatt átvihető maximális anyag mennyiségének tekinthetünk. Folyamnak nevezzük az anyag áramlásának egy olyan leírását, ami eleget tesz ezen kapacitásoknak, és a forrásoktól és nyelőktől különböző csomópontokon csak átáramlás történik, vagyis amennyi anyag beérkezik, annyi távozik is. Ekkor a folyam értéke alatt a forrásokból a nyelőkbe elszállított anyag mennyiségét értjük. Érdekes kérdés, hogy a hálózatban hogyan lehet egy maximális értékű folyamot találni. Az említett példákban értelemszerűen nulla az egyes összeköttetéseken átmenő anyag mennyiségének alsó korlátja, így triviális kérdés, hogy létezik-e egyáltalán folyam, és ekkor mi egy minimális értékű folyam. Viszont más problémáknál ezek is értelmes kérdésekké válnak.

Az egyik alkalmazási terület természetesen a tényleges hálózatok kapacitásának vizsgálata. A problémával foglalkozó egyik első cikkben [21] Ford és Fulkerson moti-

vációként egy a szovjet vasúthálózat kapacitásának felmérésével foglalkozó jelentést nevez meg [33], mely jelentés egyébként 1999-ig titkos volt [48]. A vasutak életében ma is használják a hálózati folyam modellt, például mozdonyok szerelvényekhez való rendeléséhez, személy- és teherszállítás esetén egyaránt. Utóbbi esetben különösen fontos a probléma hatékony megoldása, hiszen gyakori a közeli időpontra történő új megrendelés érkezése, vagy éppen egy megrendelés utolsó pillanatbeli lemondása, így a modellt gyakran újra kell számolni. Ezzel a problémával részletesebben is fogunk foglalkozni a dolgozatban.

A további alkalmazások között megemlíthető még a kommunikációs hálózatok megbízhatóságának elemzése, ahol a maximális éldiszjunkt utak számának meghatározásához, vagy ami ezzel ekvivalens, minimális élszámú vágás kereséséhez használják. Ezzel jól jellemezhető, hogy a hálózat bizonyos elemeinek meghibásodása esetén mennyire marad használható a rendszer. De meglepőbb területeken is felbukkan ez a modell, például a képfeldolgozás során a kép előterét és hátterét lehet elkülöníteni lényegében a pixelrács szomszédos elemei közötti hasonlóság számszerűsítésével, és a kapott hálózaton minimális vágást keresésével [32]. Egy másik példa pedig az úgynevezett mátrixkerekítési probléma, ahol adott valós számok egy táblázata, és az elemeit úgy szeretnénk felfelé vagy lefelé kerekíteni, hogy a sor- és oszlopösszegek is csak a felső vagy alsó egészrészükre változzanak. Ez megfogalmazható egy olyan hálózati folyamként, melynek az adatok egy nem egész megengedett megoldását adják. Erre alkalmazva valamely megoldó algoritmust egy egészértékű megoldást kapunk a kívánt tulajdonságokkal [7]. A hálózati folyam problémát és alkalmazásait részletesebben tárgyalja [42], [45] és [24].

1.2. A téma eddigi eredményei

A hálózati folyam problémával kiemelten először Ford és Fulkerson foglalkozott. Első cikkükben [22] bebizonyították a maximális folyamérték és a minimális értékű vágás értékének egyenlőségét, bár még nem konstruktív módon, továbbá leírtak egy algoritmust, mely egy a forrást és nyelőt összekötő él bevezetése mellett is síkbarajzolható gráfokon működik (ez nem túl korlátozó feltevés az általuk vizsgált vasúthálózatok esetén). Későbbi cikkükben [23] lényegében bevezették a ma javítóútnak nevezett objektumot, bár erre csak egy a folyamhoz rendelt mátrix manipulációjaként tekintettek, a szemléletes jelentést nem vizsgálták. Megjegyezzük, hogy itt Kőnig Dénes

és Egerváry Jenő[1] ([43], [19]) hozzárendelési feladatokra adott módszerét vették alapul, mely később magyar módszerként híresült el [44].

Az algoritmusuk vázlatosan úgy nézett ki, hogy az üres folyamból kiindulva minden lépésben keresnek egy javítóutat, és az út mentén javítanak amennyit lehet. Egész kapacitásokkal rendelkező folyam esetén világos, hogy minden javítóút mentén legalább egységnyit lehet javítani, így az élek összkapacitása egy felső korlát a szükséges javítások számára. A továbbiakban a gráf csúcsainak számát n-nel, éleinek számát m-mel, és a legnagyobb kapacitást K-val fogjuk jelölni. Így azt mondhatjuk, hogy egész kapacitásokkal rendelkező gráfon $\mathcal{O}(mK)$ javítóút szükséges a maximális folyam megkereséséhez. Elhagyva a kapacitások egészértékűségét azonban az algoritmus végessége sem garantált, sőt, még az sem, hogy az egyes javítások utáni folyamértékek sorozata a maximális folyamértékhez konvergál.

Az 1970-es évek elején ezen javított Edmonds és Karp [18], akik tetszőleges valós kapacitásokkal rendelkező hálózatra bizonyították, hogy mindig a legrövidebb javítóutat véve legfeljebb nm javítás szükséges. Ez annyin múlik, hogy sikerült belátni a javítóutak hosszának monoton növekedését (nem csökkenését), illetve azt, hogy k hosszú javítóútból egymás után legfeljebb m darabot használhatunk. Így kihasználva, hogy egy javítóút legfeljebb n hosszú lehet, adódik a korlát. Ekkor egyszerű implementációval $\mathcal{O}(nm^2)$ lépésben megvalósítható a maximális folyam keresése.

Tőlük függetlenül Dinic [17] is belátta, hogy legfeljebb nm javításra van ekkor szükség, de egy ügyes ötlettel $\mathcal{O}(n^2m)$ lépésre sikerült az algoritmusa lépésszámát csökkentenie. Egy úgynevezett szintező gráf felépítésével az összes k hosszú javítóúton egyszerre történik meg a javítás, a további lépésekben pedig már csak k-nál hosszabb javítóutak léteznek.

Egy új megközelítést jelentett az úgynevezett előfolyamokra épülő módszer. Az előfolyam a folyamnál egy gyengébb struktúra: ugyanúgy megköveteljük a kapacitások betartását, de a közbülső csúcsokra csak annyit kötünk ki, hogy a bejövő mennyiség legalább annyi legyen mint a kimenő. A módszer a javítóutas algoritmusokkal ellentétes irányból közelíti meg a problémát. Míg ott a folyam struktúrát minden lépésben megőrizve növeltük a folyam értékét a maximumig, addig az előfolyamos algoritmusoknál a forrásból kiáramló mennyiség optimális, vagy a fölötti szinten tartása mellett lépkedünk előfolyamokon, amíg a struktúra meg nem javul.

Karzanov 1974-es cikke [39] számít a módszer alapművének. A Dinic által bevezetett technikákkal ötvözve egy $\mathcal{O}(n^3)$ lépésszámú algoritmust sikerült létrehoznia.

[1]Kőnig Dénes (1884–1944) és Egerváry Jenő (1891–1958) mindketten a Budapesti Műszaki Egyetem tanárai voltak.

Egy egyszerűbb, de továbbra is $\mathcal{O}(n^3)$ leírást ad Malhotra, Kumar és Maheshwari [46], míg Cherkassky [11] illetve Galil [25] hasonló algoritmusai ritka gráfokra javítják az elméleti lépésszámot: $\mathcal{O}(n^2m^{1/2})$ és $\mathcal{O}(n^{5/3}m^{2/3})$, közel teljes gráf ($m \approx n^2$) esetén ezek is lényegében $\mathcal{O}(n^3)$ lépést jelentenek.

Az előfolyamos algoritmusokban szemléletváltást hozott Goldberg és Tarján 1988-ban megjelent cikke [29]. A csúcsoktól a nyelőig vezető javítóút távolságát becslő cimkézést vezettek be. Minden lépésben egy kisebb cimkéjű csúcs felé tol folyamot az algoritmus, illetve ha egy aktív csúcsnak (ahol több a bejövő folyam, mint a kimenő) nincs ilyen szomszédja, akkor egy újracimkézést hajt végre a csúcson. Bebizonyítható, hogy a csúcsok cimkéi nem mennek $2n$ fölé, így legfeljebb $2n^2$ cimkézés történik, továbbá FIFO (first in first out) szabállyal választva az aktív csúcsok közül legfeljebb $4n^3$ tolás után leáll az algoritmus. Mivel egy tolás műveletigénye konstans (ellentétben egy javítóúton való javításával), így ebből könnyen kijön az algoritmus egy $\mathcal{O}(n^3)$ lépésszámú implementációja. A jellemző műveletei alapján az ilyen típusú algoritmusokra "push-relabel" algoritmusként hivatkozik az angol nyelvű szakirodalom.

Goldberg és Tarján algoritmusáról is elmondható, hogy $\mathcal{O}(n^3)$ lépéssel implementálható, és Karzanov cikkéhez hasonlóan az ő cikkük is sok kutatást ihletett. Cheriyan és Maheshwari [10] megmutatta az egyik változatról, hogy $\mathcal{O}(n^2m^{1/2})$ lépésben fut, Ahuja, Orlin és Tarján ([2], [3]) különböző skálázási technikák bevezetésével pedig $\mathcal{O}(nm+n^2\log K)$, $\mathcal{O}(nm+n^2\log K/(\log\log K))$ és $\mathcal{O}(nm+n(\log K)^{1/2})$ lépésszámú algoritmusokat hoztak létre.

Ezek az előfolyamos algoritmusok már nemcsak az elméleti lépésszám, de különböző tesztfeladatokon mért futásidő tekintetében is jobban teljesítettek mint a korábbi javítóutas algoritmusok ([1], [4], [16]). Itt jegyeznénk meg, hogy az elméleti lépésszám tovább javítható speciális adatstruktúrák használatával ([50], [26], [51]), de a gyakorlatban ez sokszor az algoritmus lassulásához vezet.

Eddig a hálózati folyam probléma kombinatorikus algoritmusainak történetét foglaltuk össze, de a feladat felfogható egy speciális struktúrájú lineáris programozási feladatként, így felmerül a kérdés, hogy az általános lineáris programozási feladatra kifejlesztett algoritmusok hogyan viselkednek itt.

A szimplex módszer [14] átültetését már Dantzig is leírta [15]. Bár az így kapott módszer nagyon szép struktúrával rendelkezik, több okból mégsem vált felkapott kutatási témává. Egyrészt az általános lineáris programozási feladatra máig sem ismert polinomiális változata a szimplex algoritmusnak, és speciálisan a hálózati folyam probléma során is nagyon gyakori a degenerált pivotálás. Másrészt azon lépések

amikor mégis növekszik a folyam értéke lényegében megfelelnek egy javítóút mentén történő javításnak. Így nem meglepő, hogy inkább a kombinatorikus algoritmusok tűntek ígéretesebbnek.

Ezt a szemléletet törte meg Goldfarb és Hao 1990-ben ([30], [31]) az első polinomiális variáns publikálásával. Az algoritmus a forrástól való egyfajta távolság segítségével megcimkézi a csúcsokat, majd ezen cimkék szerint választ pivotálási pozíciót. A gráfstruktúra erős kihasználásával így sikerült bizonyítaniuk, hogy legfeljebb nm pivotálás után (mely tartalmazhat degenerált pivotokat is) megtalálnak egy optimális folyamot. Ez egyszerű implementációval $\mathcal{O}(n^2m)$ lépésszámot jelent, ami megegyezik a javítóutas algoritmusok lépésszámával. Az úgynevezett dinamikus fa adatstruktúrát [51] alkalmazva pedig a futásidő levihető $\mathcal{O}(nm \log n)$-re [28], ami már egy logaritmikus faktortól eltekintve megegyezik az előfolyamos algoritmusok lépésszámával.

A lineáris programozás másik sokat kutatott megoldási módszerei a belső pontos módszerek. Az általános lineáris programozási feladatra a 80-as évekre kifejlődő polinomiális algoritmusok ([38], [41]) még nem vehették fel a versenyt a hálózati folyamokra specializált kombinatorikus algoritmusokkal.

Az évtized végére kifejlődő előfolyamos algoritmusok hatékonysága miatt inkább csak bizonyos általánosabb hálózati folyam modellek megoldására jelentek meg belső pontos algoritmusok. Ilyen probléma például a többtermékes hálózati folyam, vagy az úgynevezett általánosított hálózati folyam, ahol az anyagnak csak egy bizonyos hányada marad meg egy élen való áthaladás során. Az 1990-es évek óta sok cikk foglalkozik ezekkel a problémákkal ([37], [47], [36]).

1.3. A diplomamunka szerkezete

A szakdolgozat témáját részben egy vasút-optimalizálási probléma ihlette. Az úgynevezett gyenge mozdony-hozzárendelési probléma során arra keresük a választ, hogy vasúti vontatási feladatok egy adott rendszerét hogyan lehet a lehető legkevesebb mozdonnyal elvégezni. A probléma "gyenge" változata alatt azt értjük, hogy figyelmen kívül hagyjuk a mozdonyok karbantartási szükségleteit, amivel együtt a probléma NP-teljessé válna. A probléma modellezési aspektusairól a témavezető [34] cikkében, illetve Barta Zsuzsanna szakdolgozatában [8] olvashatunk többet.

A probléma felírható egy hálózati folyamként, így az előző alfejezetben leírt bármely módszerrel megoldható. Az érdeklődésünket főleg a pivotalgoritmusok keltették fel, mivel a kapott gráf speciális struktúráját talán ezzel a megközelítéssel lehet

a legjobban kihasználni. A 2. fejezetet ezért a maximális folyam probléma hagyományos Ford-Fulkerson-féle felépítését követően a Dantzig nevéhez köthető hálózati szimplex keretalgoritmus ismertetésével kezdjük. Mivel a modellben az élekre vonatkozó nem nulla alsó korlátok is fellépnek, így bemutatjuk az ilyen problémák hagyományos átírását nulla alsó korlátos problémává, mely közben a modell legfeljebb kétszeresére nő.

Kutatásunkat Goldfarb és Hao eredményeinek ([30], [31]) megismerésével kezdtük. A bizonyítási technikák közelebbi vizsgálata során észrevettük, hogy az eredetileg csak nulla alsó korlátos feladatokra felírt algoritmus kisebb-nagyobb módosításokkal kiterjeszthető nem nulla alsó korlátos feladatok megoldására is. Ezek a módosítások a fízibilitási MBU algoritmus ([9], [12]) egy specializációjához vezetnek. Míg az általános lineáris programozási feladat megoldása során viszonylag nagy szabadsági fokkal választhatunk belépő változót, addig ha a hálózati folyam esetén a Goldfarb és Hao által bevezetett cimkézési technika segítségével döntünk ezen változók között, akkor az algoritmus polinomialitása is bizonyítható. Tudomásunk szerint ez az első pivotalgoritmus, amiről bizonyították nem nulla alsó korlátú hálózati folyamon a polinomialitást. A 3. fejezetben leírjuk Goldfarb és Hao algoritmusát, illetve az MBU algoritmus hálózati folyam variánsát, és bizonyítjuk a helyességüket és a polinomiális lépésszámot. Végül az algoritmust a hálózati folyam irányított gráfján és pivot táblán párhuzamosan szemléltetjük.

Elvégeztük a pivotalgoritmus számítógépes implementálását C++ nyelven [27]. Különböző méretű gyenge mozdony-hozzárendelési problémákon vizsgáltuk a programunk futásidejét, illetve az XPRESS-MP megoldójának [6] futásidejét a MOSEL nyelven felépített modellen. A probléma speciális struktúrájának kihasználásával biztató eredményeket értünk el. A 4. fejezetben röviden összefoglaljuk a gyenge mozdony-hozzárendelési probléma egy lehetséges matematikai modelljét, és ismertetjük a futási eredményeinket.

A szakdolgozatot eredményeink összefoglalásával és jövőbeli kutatási irányok kijelölésével zárjuk.

1.4. Jelölések

$G = (V, E)$	a hálózat irányított gráfja
s	a hálózat forrása
t	a hálózat nyelője
n	G csúcsainak száma
m	G éleinek száma
$G^* = (V, E^*)$	a hálózat irányított gráfja a (t, s) éllel együtt
A	G^* illeszkedési mátrixa
x^i	az $i.$ pivotálás előtti folyamértékek
l	az élek alsó korlátja
u	az élek felső korlátja
π, λ, μ	duál változók
T_i	az $i.$ pivotálás előtti feszítőfa
S_i	$T_i \backslash (t, s)$ s-et tartalmazó részgráfja
Z_i	$T_i \backslash (t, s)$ t-t tartalmazó részgráfja
C_i	az $i.$ pivotálásnál lehetséges belépő élek
p_i	az $i.$ pivotálás során belépő él
q_i	az $i.$ pivotálás során kilépő él
$d_i(v)$	a v csúcs $i.$ pivotálás előtti cimkéje
$\delta_i(e)$	az e él $i.$ pivotálás előtti cimkéje
(g, h)	a javítandó él
T_i^g	$T_i \backslash (g, h)$ g-t tartalmazó részfája
T_i^h	$T_i \backslash (g, h)$ h-t tartalmazó részfája

2. fejezet

A probléma felírása

2.1. A maximális folyam-feladat

Tekintsünk egy $G = (V, E)$ összefüggő, irányított gráfot egy kitüntetett $s \in V$ forrással, és $t \in V$ nyelővel, valamint egy $u : E \to \mathbb{R}_{\oplus}$ kapacitásfüggvénnyel.

Egy $x : E \to \mathbb{R}_{\oplus}$ függvényt megengedett folyamnak nevezünk, ha egyrészt a forráson és a nyelőn kívül minden csúcsra a beérkező és a kimenő folyam mennyisége megegyezik, valamint egyik élre sem haladja meg az él kapacitását.

Az s-ből t-be eljutó mennyiséget a folyam értékének nevezzük, ez megegyezik az s-ből kimenő, illetve a t-be beérkező folyam mennyiségével. A feladat egy maximális értékű megengedett folyam keresése.

A feladatot képletekkel leírva világossá válik, hogy egy speciális struktúrájú lineáris programozási feladatról van szó:

$$\sum_{(s,v)\in E} x(s,v) \to \max$$

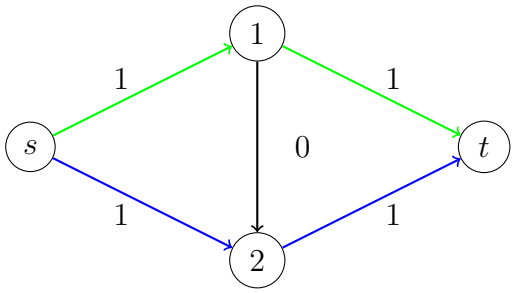
$$\forall v \in V\backslash\{s,t\}: \quad \sum_{(w,v)\in E} x(w,v) = \sum_{(v,w)\in E} x(v,w)$$

$$\forall e \in E: \quad 0 \le x(e) \le u(e)$$

Megengedett folyam mindig létezik, ilyen például az $x \equiv 0$ folyam. Természetes ötlet, hogy keressünk egy olyan $s \to t$ irányított utat, melyen nincsen telített él, ekkor egy pozitív értékkel meg tudjuk növelni a folyam értékét ezen út mentén úgy, hogy az még megengedett maradjon.

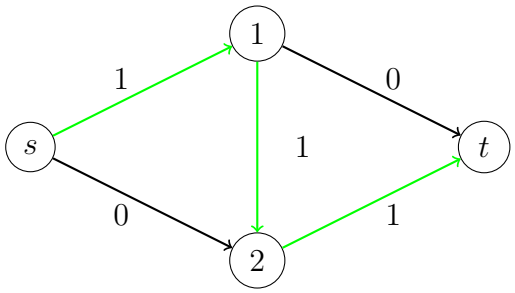
Könnyen mutatható azonban példa olyan megengedett folyamra, ahol ilyen út nem található, de a folyamérték mégsem optimális. Tekintsük a következő hálózatot: $V = \{s, 1, 2, t\}$ és $E = \{(s,1), (s,2), (1,2), (1,t), (2,t)\}$, és legyen minden él kapacitása egységnyi.

Kiindulva az azonosan nulla folyamból, majd egységnyit növelve az $s \to 1 \to t$ és az $s \to 2 \to t$ utakon optimális folyamot kapunk, hiszen két egységnél többet nem is tudunk kivinni az s csúcsból.



2.1. ábra. Optimális folyam

Ellenben ha a nulla folyamból kiindulva először az $s \to 1 \to 2 \to t$ úton növelünk egységnyit, akkor egy olyan 1 értékű folyamot kapunk, ahol már nincs $s \to t$ út telítetlen élekből.



2.2. ábra. Nem optimális folyam

Tehát valamivel összetettebb módszerre van szükségünk. Javítóút alatt egy olyan $s \to t$ irányítatlan utat értünk, melyen az előreélek telítetlenek, a visszaéleken pedig pozitív a folyam. Könnyen meggondolható, hogy ha egy javítóút előreélein növelünk, visszaélein pedig csökkentünk egységnyit, akkor továbbra is teljesülnek a csúcsokra vonatkozó megmaradási egyenletek, a folyam értéke pedig növekszik.

Az előző példában (2.2 ábra) javítóutat még találunk: $s \to 2 \to 1 \to t$. Valóban: $(s, 2)$-n növelve, $(1, 2)$-n csökkentve és $(1, t)$-n növelve egy egységnyit megkapjuk az 2.1 ábrán szereplő optimális megoldást. A következő lemma mutatja, hogy ez az objektum már tényleg megfogja a probléma lényegét.

2.1.1. Lemma. *Ha nem található javítóút a gráfban, akkor a folyam optimális.*

Bizonyítás. Egy s-et és t-t elválasztó vágás előreélei felső korlátjának összegét nevezzük a vágás értékének. Szemléletesen nyilvánvaló, hogy a hálózat egy ilyen vágásának értéke felső korlát tetszőleges megengedett folyam értékére.

Legyen S azon csúcsok halmaza, melyek elérhetők s-ből telítetlen előreéleket és pozitív értékű visszaéleket használva. Természetesen $s \in S$, és mivel nincs javítóút a gráfban, így $t \in V \backslash S$. Legyen C az $(S, V \backslash S)$ vágás. Ekkor S definíciójából adódik, hogy C előreélei telítettek, visszaélei pedig nulla értékűek, így a folyam értéke megegyezik a C vágás értékével, és így szükségképpen maximális.

□

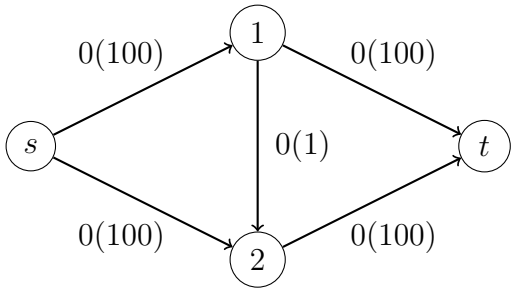
Mivel egy maximális folyam esetén nincs javítóút a gráfban, a fenti módszerrel találunk a folyam értékével egyező értékű vágást. Így bebizonyítottuk Ford és Fulkerson neves tételét:

2.1.2. Tétel (Ford, Fulkerson [22]). *Az s-ből t-be szállítható maximális folyam értéke megegyezik az s és t csúcsokat elválasztó vágások minimális értékével.*

A tétel nem csak elméleti szempontból jelentős. Az algoritmusok többsége a maximális folyam kiszámolásán túl megad egy a folyamhoz tartozó minimális vágást is, amivel nagyon nagy méretű problémák esetén is pillanatok alatt leellenőrizhetjük, hogy a kapott megoldás tényleg optimális.

A fentiekből kaphatunk egy naiv algoritmust a probléma megoldására: induljunk ki egy megengedett folyamból (mondjuk a nulla folyamból), és ismételten keressünk javítóutakat, amíg lehet. Könnyen látható, hogy egész értékű kapacitások mellett ez az algoritmus véges, hiszen minden javítás során legalább 1-et növelünk a folyam értékén.

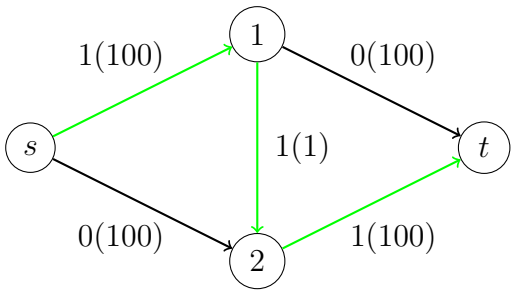
Irracionális kapacitások esetén a végesség nem feltétlenül igaz, sőt, még az sem biztos, hogy a folyamérték az optimálishoz konvergál. De egész kapacitásokra sem feltétlenül lesz polinomiális az algoritmus, mint azt a következő példán láthatjuk (2.3. ábra).



2.3. ábra. Nem polinomiális példa – hálózat

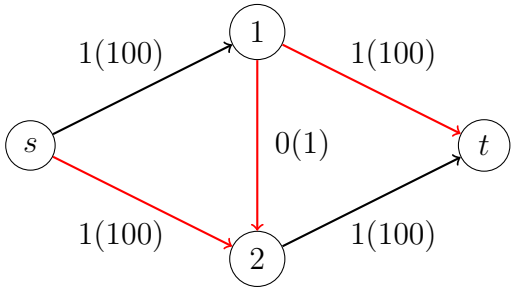
A maximális folyam értéke 200, hiszen ilyen megengedett folyamot találunk, és az $\{(s,1),(s,2)\}$ vágás értéke is 200. Ezt a folyamot megkaphatjuk mindössze két javítással, először az $s \to 1 \to t$ úton javítva 100-at, majd az $s \to 2 \to t$ úton javítva 100-at.

Választhatjuk viszont első javítóútnak is a hosszabb $s \to 1 \to 2 \to t$ utat, itt az $(1,2)$ él kapacitása miatt csak 1-et tudunk javítani (2.4. ábra).



2.4. ábra. Nem polinomiális példa – 1. lépés

Ezután pedig választhatjuk az $s \to 2 \to 1 \to t$ javítóutat, ahol ismét az $(1,2)$ él miatt csak egyet tudunk javítani (2.5. ábra).



2.5. ábra. Nem polinomiális példa – 2. lépés

Könnyen látható, hogy ha ezt a két javítóutat használjuk felváltva, akkor a folyam értéke csak 200 lépésben éri el a maximumot. Továbbá 100 helyett nagyobb kapacitást használva a megfelelő éleken még több lépést végeznénk ezen a csupán 4 csúcsból és 5 élből álló hálózaton.

Ezt a problémát oldotta meg nyugaton Edmonds és Karp [18], valamint tőlük függetlenül keleten Dinic [17], amikor belátták, hogy mindig az egyik legrövidebb javítóutat használva legfeljebb nm javításra van szükség, tetszőleges valós kapacitások esetén.

Érdemes még megemlíteni a javítóutas módszer egy az alkalmazások szempontjából fontos következményét. Ha minden kapacitás egész értékű, akkor létezik olyan

maximális folyam, mely csak egész értékeket vesz fel, hiszen például Dinic algoritmusát használva egész értékű folyamokon keresztül jutunk el véges időben egy maximális folyamhoz.

2.2. Maximális folyam mint LP-feladat

Az LP feladat felírását gyakorlatilag már elvégeztük a fejezet elején. Az egyszerűbb kezelés érdekében a folyam feladatot cirkulációs feladattá alakítjuk egy $t \to s$ él bevezetésével.

Legyen tehát $E^* := E \cup (t,s)$ és $G^* := (V, E^*)$. Ekkor a csúcsokra vonatkozó megmaradási egyenletek egységesebbé válnak, az s és t csúcsra is ugyanolyan alakúak lesznek, mint a többi csúcsra. A célfüggvény egyszerűsödik: az s-ből t-be eljuttatott folyam értéke megegyezik a (t,s) élen visszahozott "folyammal". Tekintsük továbbá a nem nulla alsó korlátokkal rendelkező általánosabb feladatot:

$$\begin{aligned} & & \max x_{t,s} & \\ \forall v \in V: & & \sum_{(w,v)\in E^*} x_{w,v} - \sum_{(v,w)\in E^*} x_{v,w} &= 0 \\ \forall e \in E: & & l_e \le x_e &\le u_e \end{aligned} \tag{2.1}$$

A csúcsokra vonatkozó egyenletek röviden $A\mathbf{x} = \mathbf{0}$ alakba írhatók, ahol A a G^* gráf illeszkedési mátrixa. Ismert, hogy egy n csúcsú összefüggő gráf illeszkedési mátrixának rangja $n-1$, továbbá a mátrix valamely $n-1$ oszlopa pontosan akkor lineárisan független, ha az oszlopoknak megfelelő élek feszítőfát alkotnak a gráfban [40].

Következésképpen (2.1) egy B bázisa megfelel G^* egy T feszítőfájának, mely tartalmazza a (t,s) élet, hiszen a hozzá tartozó változó nem korlátos. A B-hez tartozó bázismegoldásban a bázison kívüli változók értéke megegyezik alsó vagy felső korlátjukkal, vagyis a T-n kívüli élek folyamértéke megegyezik az alsó vagy felső korlátjukkal.

Továbbá azt is tudjuk egy irányított gráf illeszkedési mátrixáról, hogy teljesen unimoduláris, vagyis minden négyzetes részmátrixának determinánsa 0, 1 vagy -1. Ebből egész jobboldal esetén következik, hogy a lineáris programozási feladat bármely bázismegoldása egészértékű. Ugyanis tetszőleges B bázisra A_B^{-1} egészértékű, mint azt az inverzképzés adjungáltas képletéből láthatjuk: $A_B^{-1} = \mathrm{adj} A_B / \det A_B$.

Primál megengedett bázismegoldásnak egy olyan bázismegoldást nevezünk, ahol az $l_e \le x_e \le u_e$ korlátok is teljesülnek. A bázison kívüli változókra ezek automatikusan teljesülnek, vagyis primál nem megengedett változó csak a bázisban lehet.

Egy adott T feszítőfából elhagyva a (t,s) élet a feszítőfa két összefüggő részre esik, jelöljük S-sel az s csúcsot tartalmazó részfát, és Z-vel a t csúcsot tartalmazót. Egy bázismegoldás duál megengedett, ha az $S \to Z$ élekre $x_e = u_e$, míg a $Z \to S$ élekre $x_e = l_e$. Ennek belátásához írjuk fel a (2.1) feladat duálisát:

$$\begin{aligned} & \min \sum_{e \in E} u_e \lambda_e - l_e \mu_e \\ \forall (v,w) \in E: \quad & \pi(w) - \pi(v) + \lambda_{v,w} - \mu_{v,w} \geq 0 \\ & \pi(s) - \pi(t) \geq 1 \\ \forall e \in E: \quad & \lambda_e, \mu_e \geq 0 \end{aligned} \tag{2.2}$$

Ekkor csak annyit kell meggondolnunk, hogy a duális feladat következő megoldása megengedett, valamint a primál feladat fenti megoldása komplementáris az eltérés változókra:

$$\begin{aligned} \pi(v) &= \begin{cases} 1 & \text{ha } v \in S, \\ 0 & \text{egyébként.} \end{cases} \\ \lambda_{v,w} &= \begin{cases} 1 & \text{ha } v \in S \text{ és } w \in Z, \\ 0 & \text{egyébként.} \end{cases} \\ \mu_{v,w} &= \begin{cases} 1 & \text{ha } v \in Z \text{ és } w \in S, \\ 0 & \text{egyébként.} \end{cases} \end{aligned} \tag{2.3}$$

Így egy adott T feszítőfához tartozó bázismegoldásban a duál nem megengedett változók azon $S \to Z$ élekhez tartozó változók, melyekre $x_e = l_e$, és azon $Z \to S$ élekhez tartozó változók, melyekre $x_e = u_e$. A kombinatorikus szemlélettel is könnyen látható, hogy egy primál és duál megengedett bázismegoldás optimális, hiszen a primál megengedettség miatt $\mathbf{x}$ megengedett folyam, míg a duál megengedettség miatt az (S,Z) vágás értéke megegyezik a folyam értékével.

A (2.1) és (2.2) lineáris programozási feladatpár együtt a következő lineáris komplementaritási feladatként írható fel (ismét A-val jelölve G^* illeszkedési mátrixát):

$$\begin{aligned} A\mathbf{x} = 0 \qquad & -A^T \pi + \begin{pmatrix} \lambda \\ 0 \end{pmatrix} - \begin{pmatrix} \mu \\ 0 \end{pmatrix} \geq \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \qquad & \lambda, \mu \geq 0 \\ \forall e \in E: \qquad & \lambda_e (u_e - x_e) = 0, \qquad \mu_e (x_e - l_e) = 0 \end{aligned} \tag{2.4}$$

2.3. Pivotálás

A lineáris programozási feladat pivotalgoritmussal történő megoldása során bázismegoldásról bázismegoldásra lépünk, egy ilyen lépést nevezünk pivotálásnak. A pivotáláshoz ki kell választanunk egy a bázisba belépő, és egy onnan kilépő változót. Egy adott bázis esetén persze nem választhatjuk bármelyik változópárt, hiszen a változócsere után is bázist szeretnénk kapni. A folyamprobléma nyelvén feszítőfáról feszítőfára szeretnénk lépni.

Ha egy adott T feszítőfa esetén először egy p belépő élet választottunk, akkor a kilépő q élet a $T \cup \{p\}$-ben egyértelműen létrejövő kör éleiből kell kiválasztanunk, hogy ismét körmentes részgráfot kapjunk. Míg ha először egy q kilépő élet választunk, akkor a $T \backslash \{q\}$ feszítőfa két részfára esik, és a belépő p élet az általuk meghatározott vágásból kell választanunk, hogy ismét összefüggő részgráfot kapjunk.

A pivotálás során a $T \cup \{p\}$-ben kialakuló kör folyamértékei változnak. A kör valamelyik irányítása szerinti előreéleken növeljük d-vel a folyamértéket, míg a visszaéleken csökkentjük d-vel a folyamértéket, ahol a d mennyiséget úgy kell megválasztanunk, hogy a kilépő q él új folyamértéke l_q vagy u_q legyen. Látható, hogy ekkor a csúcsokba beérkező és onnan kiáramló folyam mennyisége továbbra is egyenlő marad.

A Goldfarb és Hao által leírt algoritmus a hálózati primál szimplex algoritmus egy variánsa, így érdemes ezt a keretalgoritmust külön megvizsgálnunk. A pivotálás során először belépő élet választunk a célfüggvény növelésének céljából, majd a kilépő változó választásánál a primál megengedettség fenntartását tartjuk szem előtt.

A folyam értékének módosításához úgy kell belépő élet választanunk, hogy a kialakuló körben szerepeljen a (t, s) él is. Ehhez csak annyi kell, hogy az (S, Z) vágásból válasszunk. Továbbá mivel növelni szeretnénk $x_{t,s}$ értékét, így a belépő él $S \to Z$ alsó korláton levő él, vagy $Z \to S$ felső korláton levő él kell, hogy legyen. Úgy is fogalmazhatunk, hogy egy duál nem megengedett változónak kell belépnie a bázisba. Jelöljük ezeket a lehetséges belépő éleket C-vel:

$$\begin{aligned} C := \{(v, w) \in E : & v \in S, w \in Z, x_{v,w} = l_{v,w} \\ & v \in Z, w \in S, x_{v,w} = u_{v,w}\} \end{aligned} \tag{2.5}$$

A kialakuló P kört (t, s)-sel egyező irányítással tekintve legyen

$$\begin{aligned} f := \min\{ & u_e - x_e : e \text{ primál megengedett előreél } P\text{-ben}, \\ & x_e - l_e : e \text{ primál megengedett hátraél } P\text{-ben}\} \end{aligned} \tag{2.6}$$

P előreélein növeljünk, visszaélein csökkentsünk f-et. Kilépő élnek válasszunk egy olyan q élt, ahol az f meghatározásához használt minimum felvétetett. Ekkor

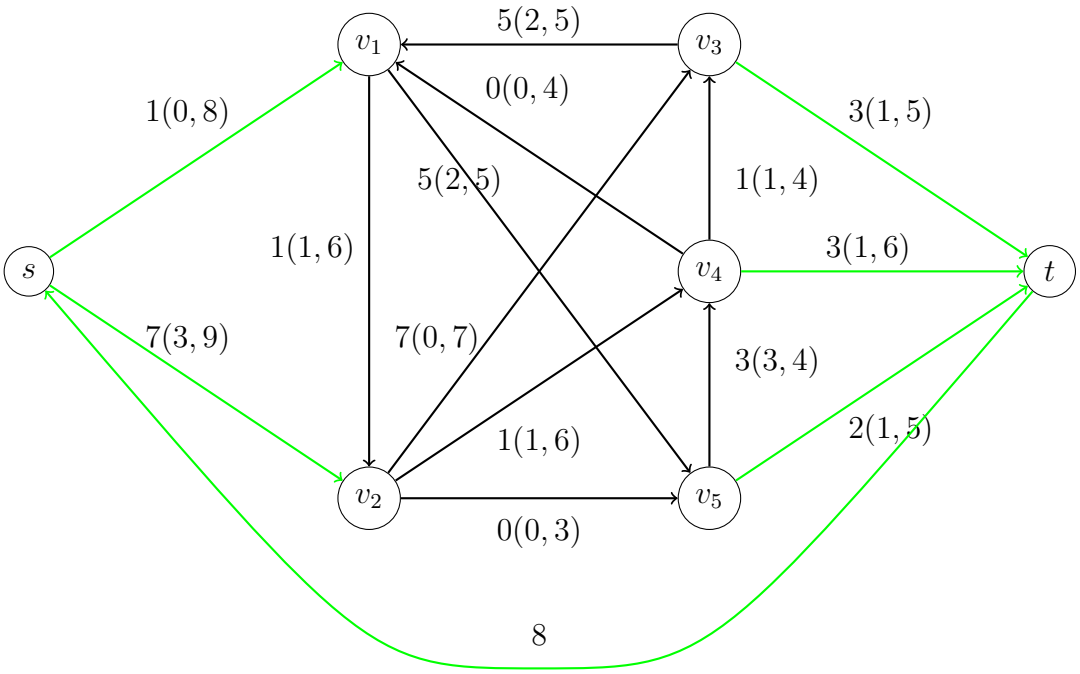
d definíciójából adódóan a primál megengedett élek megengedettek maradtak, és a kilépő él folyamértéke valamelyik korlátjával megegyezik. A kilépő él választási szabálya megegyezik az általános lineáris programozási feladatnál alkalmazott primál hányadosteszttel, mivel a pivot tábla minden eleme 0 vagy ± 1, így a hányadosok megegyeznek a folyamérték megfelelő korláttól való távolságával.

Röviden összefoglalva tehát a következőképpen néz ki a hálózati primál szimplex algoritmus:

Algoritmus 1 Hálózati primál szimplex algoritmus

x egy megengedett bázismegoldás
while $C \neq \emptyset$ **do**
  $p \in C$ tetszőleges él belép a bázisba
  meghatározzuk f-et
  javítunk a körön
  meghatározzuk a bázisból kilépő q élt
end while

Illusztrálásképp tekintsük a következő hálózati folyam feladatot (2.6. ábra). A bázisbeli élek zölddel vannak jelölve, az élek fölött az aktuális folyamérték, illetve az alsó és felső korlát található.

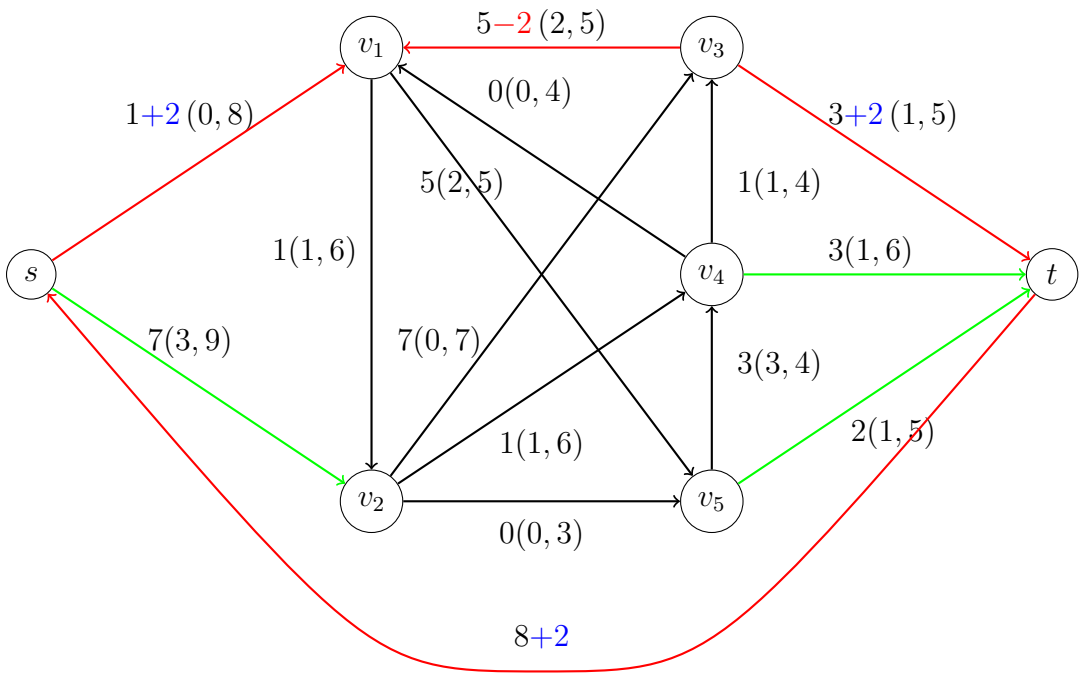


2.6. ábra. Primál pivotálás – 1

A (t, s) élt elhagyva a feszítőfa az $S = \{s, v_1, v_2\}$ és a $Z = \{v_3, v_4, v_5, t\}$ részfákra

esik. Belépő élnek választhatunk S-ből Z-be menő, alsó korláton levő élt: ilyen (v_2, v_4) és (v_2, v_5). Illetve választhatunk Z-ből S-be menő felső korláton levő élt, ilyen (v_3, v_1).

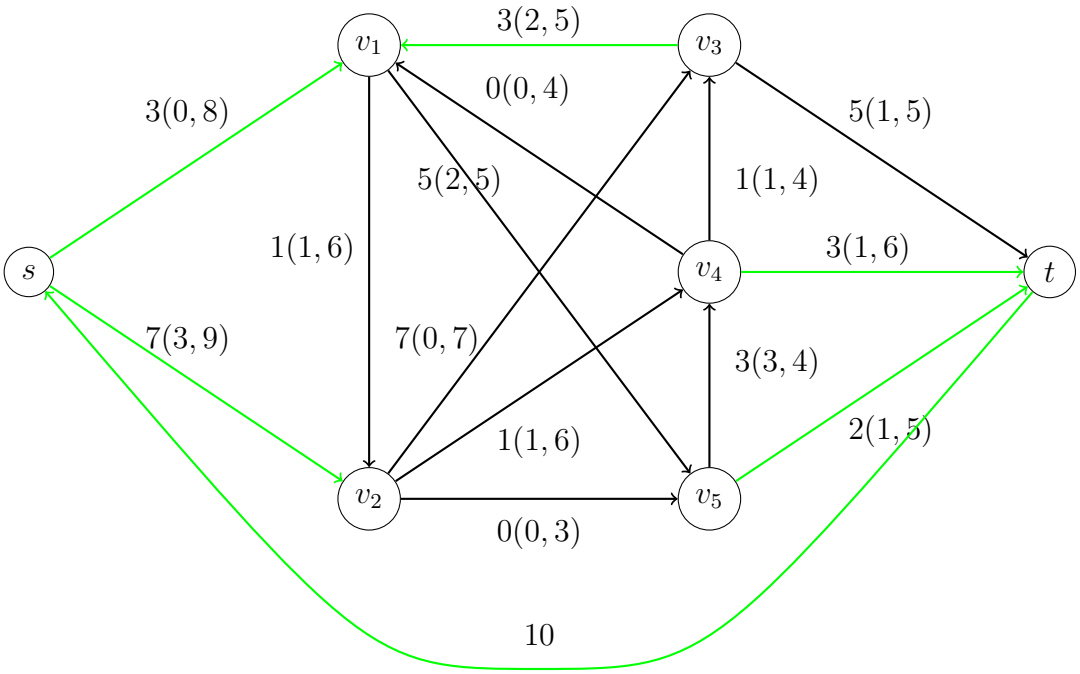
Válasszuk mondjuk (v_3, v_1)-et. A bázishoz hozzávéve (v_3, v_1)-et kialakul az (s, v_1, v_3, t) kör. Az $s \to v_1$ élen legfeljebb 7-et növelhetünk, a $v_1 \leftarrow v_3$ élen legfeljebb 3-at csökkenthetünk, a $v_3 \to t$ élen pedig legfeljebb 2-t növelhetünk. Ezek közül a 2 a legkisebb, így ennyit javíthatunk a körön az élek primál megengedettségének megőrzése mellett.



2.7. ábra. Primál pivotálás – 2

Mivel (v_3, t) miatt nem tudtunk többet javítani, így ő lép ki a bázisból, a pivotálás utáni állapotot a megváltozott bázis szerkezettel az 2.8 ábra mutatja.

Általános (nem feltétlenül hálózati folyamból származó) lineáris programozási feladatok elméletében ismert, hogy primál megengedett bázismegoldásból indulva csak primál pivotálást használva eljuthatunk az optimális megoldáshoz. A primál szimplex módszer végessége, vagyis a ciklizálás elkerülése biztosítható a pivotálási pozíció valamilyen rendszer szerinti megválasztásával (Bland-szabály, lexikografikus szimplex módszer, s-monoton index szabály [13]).



2.8. ábra. Primál pivotálás – 3

Maximális folyam keresése esetén azonban ennél több is mondható. Goldfarb és Hao cimkézési módszerével választva a belépő éleket az algoritmus polinomialitása is biztosítható, mint azt a 3. fejezetben látni fogjuk. A másik feltevés, miszerint primál megengedett bázisból indul az algoritmus egyszerűen teljesíthető a nulla alsó korlátos esetben, hiszen ekkor a nulla folyam ilyen (tetszőleges feszítőfa mellett). Nem nulla alsó korláttal rendelkező feladat esetén viszont primál megengedett bázisfolyam keresése nemtriviális feladat, az is lehetséges, hogy ilyen nem létezik.

A következő részben bemutatjuk a feladat beágyazását egy nulla alsó korlátú folyamba, melynek megoldásával megkapjuk az eredeti feladat egy megengedett bázisát (ha van ilyen). A 3. fejezetben pedig bemutatunk egy új módszert megengedett bázismegoldás keresésére, mely végig az eredeti gráfon dolgozik.

2.4. Megengedett bázisfolyam előállítása a gráf kibővítésével

Természetes ötlet bevezetni az $x'_e = x_e - l_e$ változókat (és legyen $x'_{t,s} = x_{t,s}$), hiszen így az x' változókra nulla alsó korlát fog vonatkozni. A csúcsokra vonatkozó

egyenletek azonban megváltoznak:

$$\forall v \in V : \sum_{(w,v)\in E^*} x'_{w,v} - \sum_{(v,w)\in E^*} x'_{v,w} = \sum_{(v,w)\in E^*} l_{v,w} - \sum_{(w,v)\in E^*} l_{w,v} \tag{2.7}$$

Így (2.7) jobboldalát $b(v)$-vel jelölve a következő lineáris programozási feladatot kapjuk:

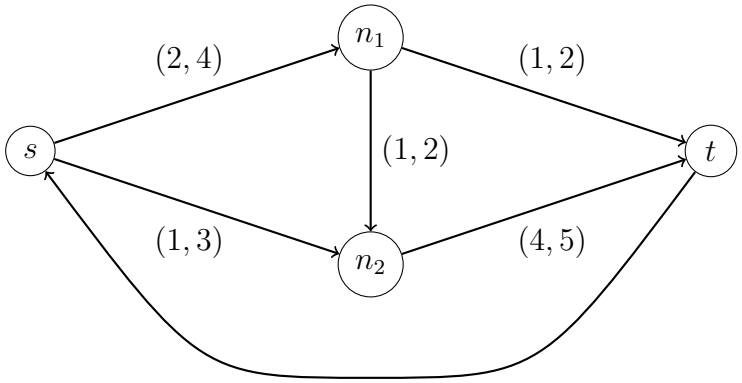
$$\begin{aligned} & \max x'_{t,s} \\ \forall v \in V : \quad & \sum_{(w,v)\in E^*} x'_{w,v} - \sum_{(v,w)\in E^*} x'_{v,w} = b(v) \\ \forall e \in E : \quad & 0 \le x'_e \le u_e - l_e \end{aligned} \tag{2.8}$$

Mivel a csúcsokra vonatkozó egyenletek jobboldala nem nulla így már nem folyam feladatról van szó, hanem egy cirkulációs feladatról ($\sum_{v\in V} b(v) = 0$). A cirkulációs feladatok elméletéből ismert a következő módszer megengedett cirkuláció előállítására.

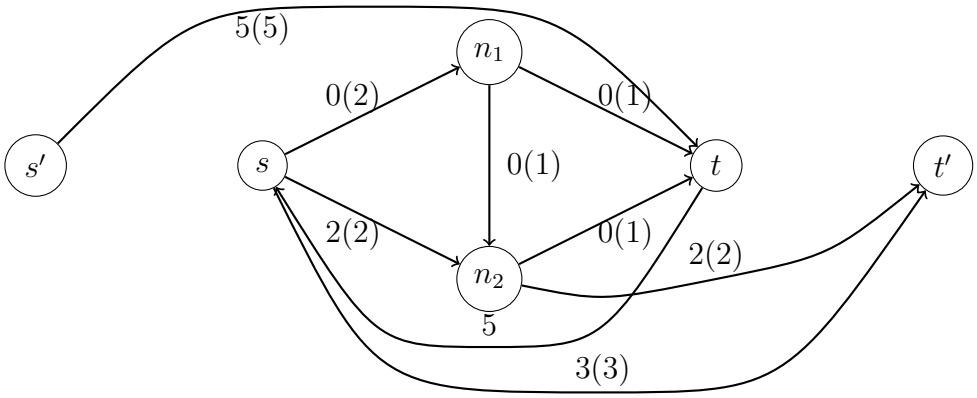
Vezessünk be két új csúcsot, jelölje őket s' és t'. Húzzuk be az (s', v) élet, ha $b(v) < 0$, és legyen ezen él felső korlátja $-b(v)$. Hasonlóan, húzzuk be a (v, t') élet, ha $b(v) > 0$, és legyen ezen él felső krolátja $b(v)$. Az új élek alsó korlátja mindkét esetben legyen 0.

Könnyen látható, hogy (2.8)-nek pontosan akkor van megengedett megoldása, ha ebben az új gráfban a maximális folyam s'-ből t'-be telíti az új éleket, és ekkor ezt a megengedett megoldást megkaphatjuk egyszerűen az új csúcsok és élek törlésével. Ezután visszacserélve az x' változókat az x változókra megkapjuk az eredeti feladat egy megengedett megoldását.

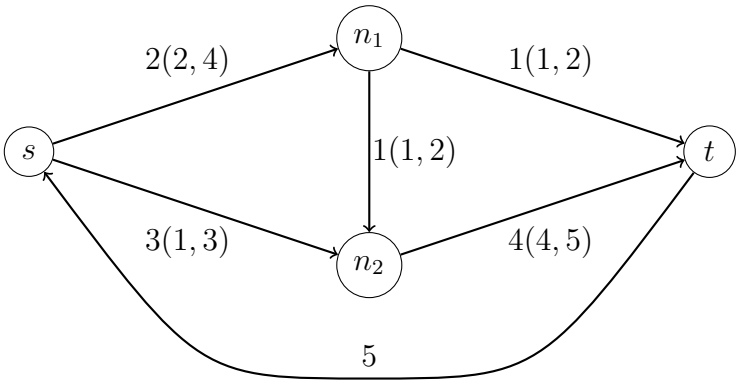
Példaként tekintsük az alábbi hálózatot. Az éleken az alsó és felső korlátok szerepelnek, a $t \to s$ él értékét szeretnénk maximalizálni.



2.9. ábra. Az eredeti hálózat



2.10. ábra. Az átalakított hálózat



2.11. ábra. Egy megengedett megoldás

Minden csúcsra kiszámoljuk $b(v)$-t: $b(s) = 3$, $b(n_1) = 0$, $b(n_2) = 2$ és $b(t) = -5$. Bevezetünk egy s' és egy t' csúcsot. Felveszünk egy $s' \to t$ élet 5 korláttal, valamint egy $s \to t'$ élet 3 korláttal és egy $n_2 \to t'$ élet 2 korláttal. A már meglevő élek felső korlátja legyen $u_e - l_e$, és minden él alsó korlátja 0. A kapott problémát meg tudjuk oldani a nulla folyamból kiindulva. A kapott hálózatot és egy optimális megoldást láthatunk az 2.10 ábrán.

Ebből elhagyva az új éleket, valamint a folyamértékekhez hozzáadva az élek alsó korlátját egy megengedett megoldást kapunk az eredeti feladatra (2.11 ábra).

3. fejezet

Algoritmusok

3.1. Goldfarb és Hao algoritmusa

A nulla alsó korlátos folyam-feladatra Goldfarb és Hao ([30] és [31]) adtak először erősen polinomiális futásidejű pivot algoritmust. Tekintsük tehát a következő feladatot:

$$\begin{aligned} & \max x_{t,s} \\ \forall v \in V: \quad & \sum_{(w,v)\in E^*} x_{w,v} - \sum_{(v,w)\in E^*} x_{v,w} = 0 \\ \forall e \in E: \quad & 0 \le x_e \le u_e \end{aligned}$$

Jelölje az i. pivotálás előtti bázist T_i, az i. pivotálás előtti folyamot pedig x^i. Kiindulásként legyen $x^1 \equiv 0$, és T_1 egy tetszőleges feszítőfa, mely tartalmazza a (t,s) élt. A T_i feszítőfából elhagyva a (t,s) élet az két feszítőfára esik, az s-et tartalmazót jelölje S_i, a t csúcsot tartalmazót pedig Z_i.

Az algoritmus egy primál szimplex algoritmus, vagyis minden lépésben primál pivotálást használ, így a változók primál megengedettségének megtartása mellett növeli a célfüggvényt, amíg el nem éri az optimumot. Az i. pivotálásnál belépő él a

$$\begin{aligned} C_i = \{(v,w) \in E: \quad & v \in S_i, w \in Z_i, x_{v,w} = 0 \text{ vagy} \\ & v \in Z_i, w \in S_i, x_{v,w} = u_{v,w}\} \end{aligned}$$

halmazból kerül ki. A kialakuló körön a (t,s) él irányításával egyező éleken növelünk, az ellentétes irányú éleken csökkentünk amennyit lehet. Végül egy olyan él lép ki, amely akadályozta a kör mentén történő további javítást.

Az algoritmus megáll, ha C_i üres valamelyik lépésben. Ekkor könnyen látható, hogy az (S_i, Z_i) vágás értéke megegyezik a folyam értékével, így az optimális.

Eddig tulajdonképpen egy általános primál szimplex algoritmust írtunk le. Ami megkülönbözteti Goldfarb és Hao algoritmusát, és ami biztosítja a polinomialitást, az a C_i élei közüli választás módszere. Tényleg sok múlik ezen a döntésen, például Goldfarb és Hao ([31]) megmutatják, hogy mindig egy olyan élet választva C_i-ből, amire a kialakuló kör a lehető legrövidebb egy nem polinomiális algoritmushoz vezet, noha a legrövidebb javítóútat használó kombinatorikus algoritmus polinomiális.

A választás az s csúcstól való egyfajta távolságon alapul. Először egy v csúcs cimkéjét definiáljuk, mint a legrövidebb $s \to v$ pszeudo-javítóút hossza:

3.1.1. Definíció (pszeudo-javítóút). *Egy T_i bázishoz tartozó x^i folyam melletti $v \to w$ pszeudo-javítóúton egy olyan $v \to w$ utat értünk, mely báziséleket tetszőleges irányban, $x_e = l_e$ bázison kívüli éleket előre irányban, illetve $x_e = u_e$ bázison kívüli éleket visszafele irányban használ.*

Algoritmikus szempontból ez egy egyszerű rekurziót jelent. Jelölje $d_i(v)$ a v csúcs i. pivotálás előtti cimkéjét, ekkor a cimkék a következőképpen számolhatók:

Algoritmus 2 Cimkézés

$d_i(s) = 0$, lista $= \{s\}$
while lista $\neq \emptyset$ **do**
  v legyen a lista első eleme
  for all $(v, w) \in E$, $((v, w) \in T_i$ vagy $x_{v,w} = l_{v,w})$, w cimkézetlen **do**
    $d_i(w) = d_i(v) + 1$
    w a lista végére kerül
  end for
  for all $(w, v) \in E$, $((w, v) \in T_i$ vagy $x_{w,v} = u_{w,v})$, w cimkézetlen **do**
    $d_i(w) = d_i(v) + 1$
    w a lista végére kerül
  end for
  töröljük v-t a listáról
end while

Gráfelméletileg ezt úgy fogalmazhatjuk meg, hogy képezünk egy $G' = (V, E')$ segédgráfot. Ha (v, w) bázisbeli él, akkor (v, w) és (w, v) is eleme E'-nek. Ha (v, w) nincs a bázisban és $x_{v,w} = 0$, akkor $(v, w) \in E'$. Illetve ha (v, w) nincs a bázisban és $x_{v,w} = u_{v,w}$, akkor $(w, v) \in E'$. Ekkor egy v csúcs cimkéje az eredeti gráfban megegyezik a legrövidebb $s \to v$ irányított út hosszával G'-ben. Ez hatékonyan számolható például egy szélességi kereséssel, a fenti cimkézési algoritmus tulajdonképpen ezzel egyezik meg.

Ezután definiálhatjuk az élek cimkéit. Jelölje egy (v, w) él i. pivotálás előtti cimkéjét $\delta_i(v, w)$. Ha $(v, w) \notin T_i$, akkor legyen

$$\delta_i(v,w) = \begin{cases} d_i(v) & \text{ha } x^i_{v,w} = l_{v,w} \\ d_i(w) & \text{ha } x^i_{v,w} = u_{v,w} \end{cases}$$

Ha $(v, w) \in T_i$ és $i \geq 2$, akkor leyen $\delta_i(v, w) = \delta_{i-1}(v, w)$, illetve minden $(v, w) \in T_1$-re legyen $\delta_1(v, w) = 0$.

Megjegyezzük, hogy az algoritmus implementálásánál csak a C_i-beli élek cimkéje lényeges, ami pedig az S_i-beli végpontjuk cimkéjeként definiálható. Így az élek cimkéit nem szükséges ténylegesen kiszámolni, inkább a polinomialitás bizonyítását teszik egyszerűbbé.

Most, hogy a szükséges fogalmakat definiáltuk, kimondhatjuk Goldfarb és Hao eredményét tételként:

3.1.2. Tétel (Goldfarb, Hao [31]). *Ha az i. pivotálás során C_i egyik legkisebb cimkéjű élét választjuk belépő élnek, akkor az algoritmus legfeljebb nm pivotálás után véget ér.*

A tétel bizonyításához először belátunk egy lemmát, mely szerint a csúcsok cimkéi bizonyos szempontból mutatják az algoritmus haladását az optimális megoldás felé:

3.1.3. Lemma. *A csúcsok cimkéi monoton növekednek, vagyis minden $z \in V$-re, és minden i-re $d_i(z) \leq d_{i+1}(z)$.*

Bizonyítás: Legyen az i. pivotálásnál belépő él $p = (v, w)$ és $x^i_p = 0$ (vagyis $v \in S_i$ és $w \in Z_i$). A másik eset hasonlóan bizonyítható.

Figyeljük meg, hogy $d_i(w) = d_i(v) + 1$: egyrészt a cimkézés definíciója alapján $d_i(w) \leq d_i(v) + 1$, másrészt ha $d_i(w) \leq d_i(v)$ lenne, akkor a w csúcs cimkézését visszakövetve egy olyan $s \to w$ pszeudo-javítóutat kapnánk, melynek van legalább egy C_i-beli éle, ennek a C_i-beli élnek pedig kisebb lenne a cimkéje mint p-nek, ami ellentmond p választásának.

A $T_i \cap T_{i+1}$-beli éleket mindkét cimkézésnél mindkét irányban lehet használni, a $\overline{T_i \cup T_{i+1}}$ éleknek pedig nem változott a folyamértéke, így mindkét cimkézésnél ugyanabban az irányban lehet őket használni.

Ha a kilépő él is p, akkor p-t az i. cimkék meghatározásánál csak előreélként, míg az $i + 1$. cimkék meghatározásánál csak hátraélként lehet használni. Vagyis

egy z csúcs cimkéje csak akkor csökkenhetett, ha az $i+1$. pivotálás előtti állapot szerinti legrövidebb $s \to z$ pszeudó-javítóút visszaélként használja p-t. De ekkor

$$\begin{aligned} d_{i+1}(z) = d_{i+1}(w) + 1 + d_{i+1}(v,z) &\geq d_i(w) + 1 + d_i(v,z) \\ &= d_i(v) + 2 + d_i(v,z) \geq d_i(z) + 2 \end{aligned}$$

ahol $d_j(v,z)$ a legrövidebb x^j melletti $v \to z$ pszeudó-javítóút hossza. A lépések indoklásai rendre:

1. Megállapítottuk, hogy a legrövidebb x^{i+1} melletti $s \to z$ pszeudó-javítóút $s \to w \xrightarrow{p} v \to z$ alakú.

2. A legrövidebb x^{i+1} melletti $s \to w$ és $v \to z$ pszeudó-javítóutak nem használhatják visszafele a (v,w) élet, így ezek az utak legalább olyan hosszúak mint x^i melletti párjuk.

3. Itt felhasználjuk, hogy $d_i(w) = d_i(v) + 1$.

4. A legrövidebb pszeudó-javítóutakra is érvényes a háromszög egyenlőtlenség: $d_i(z) \leq d_i(v) + d_i(v,z)$.

Az egyenlőtlenség két végét tekintve, ellentmondást kaptunk azzal, hogy z cimkéje csökkent.

Most tegyük fel, hogy a kilépő q él nem azonos p-vel. Ekkor p-t az i. cimkék meghatározásánál csak előre irányban, míg az $i+1$. cimkékhez mindkét irányban használhatjuk. A kilépő q élet pedig az i. cimkék meghatározásnál mindkét irányban, míg az $i+1$. cimkékhez csak az egyik irányban tudjuk használni.

Ekkor egy z csúcs cimkéje csak akkor csökkenhetett, ha a legrövidebb x^{i+1} melletti $s \to z$ pszeudó-javítóút visszaélként használja p-t. De ekkor az előző esettel egyező indoklással ismét ellentmondást kapunk:

$$\begin{aligned} d_{i+1}(z) = d_{i+1}(w) + 1 + d_{i+1}(v,z) &\geq d_i(w) + 1 + d_i(v,z) \\ &= d_i(v) + 2 + d_i(v,z) \geq d_i(z) + 2 \end{aligned}$$

Ezzel a lemma bizonyítását elvégeztük.

□

Könnyen látható, hogy a csúcsok cimkéinek monotonitásából következik az élek cimkéinek monotonitása. Az élek cimkéivel pedig már bevezethető egy olyan mennyiség, amely a pivotálások során "általában" csökken, és csak "kevésszer" nő. Az

utóbbi eset előfordulásainak számát becsülve kapható meg a tételben szereplő korlát a lépések számára.

A 3.1.2 tétel bizonyítása: Legyen az i. pivotálásnál belépő él p_i, a kilépő él q_i, és q_i Z_{i+1}-beli végpontja z. Legyen továbbá $R_i = T_i \cup p_i$ és $Q_i = \sum_{e \in R_i} \delta_e^i$.

Ekkor egyszerűen kapjuk, hogy

$$\begin{aligned}\delta_{p_{i+1}}^{i+1} = \min\{\delta_e^{i+1} \,:\, e \in C_{i+1}\} &= \min\{d_{i+1}(w) \,:\, w \in Z_{i+1}\} - 1 \\ &\leq d_{i+1}(z) - 1 = \delta_{q_i}^{i+1} - 1\end{aligned}$$

Felhasználva, hogy a bázisbeli élek cimkéje definíció szerint nem változik, illetve az előző egyenlőtlenséget:

$$Q_{i+1} - Q_i = \delta_{p_{i+1}}^{i+1} - \delta_{q_i}^i \leq \delta_{q_i}^{i+1} - \delta_{q_i}^i - 1 \tag{3.1}$$

Mivel a csúcsok cimkéi nem csökkennek, így az élek cimkéi sem, ezért két eset lehetséges:

a) $\delta_{q_i}^{i+1} > \delta_{q_i}^i$

b) $\delta_{q_i}^{i+1} = \delta_{q_i}^i$, ekkor $Q_{i+1} \leq Q_i - 1$.

Jelölje egy e élre k_e, hogy hányszor történik meg vele az a) eset. Mivel egy él cimkéje legfeljebb n, így $k_e \leq n$. (3.1) jobboldala összegezve az e él előfordulásaira: $\delta_e^f - \delta_e^1 - k_e \leq n - k_e$, ahol δ_e^f az e él utolsó cimkéje.

Így az a) esetbeli pivotálások legfeljebb $\sum_{e \in E}(n - k_e)$-vel növelhetik Q értékét az algoritmus során. Mivel $Q \geq 0$, és a b) esetben legalább 1-gyel csökken Q értéke, így a b) eset legfeljebb $\sum_{e \in E}(n - k_e) + Q_1 - Q_f$-szer történhet meg.

Így az összes pivotálás száma felülről becsülhető:

$$\sum_{e \in E'} k_e + \sum_{e \in E'}(n - k_e) + Q_1 - Q_f = nm + Q_1 - Q_f \leq nm$$

ahol az utolsó egyenlőtlenséghez elég meggondolni, hogy $Q_1 = \delta_{p_1}^1$, és R_f tartalmaz legalább egy r élet az eredeti (S_1, Z_1) vágásból, amire p_1 választásából és a cimkék monotonitásából $\delta_{p_1}^1 \leq \delta_r^1 \leq \delta_r^f$, így tényleg $Q_1 \leq Q_f$.

Ezzel a bizonyítást befejeztük.

□

Mivel a cimkék egyszeri teljes kiszámolása $\mathcal{O}(m)$ időt vesz igénybe, így minden pivotálás előtt kiszámolva a cimkéket legalább $\mathcal{O}(nm^2)$ lépést tenne az algoritmusunk. Szerencsére azonban ezt nem szükséges minden pivotálás előtt megtenni.

A Goldfarb-Hao algoritmus egy lehetséges implementációja a következőképpen néz ki:

Algoritmus 3 Goldfarb-Hao

T_1 tetszőleges feszítőfa, $x^1 \equiv 0$, $i = 1$.
Cimkézés
while $d_i(t) \neq \infty$ **do**
  keressünk egy minimális cimkéjű C_i-beli élt
  javítsunk a kialakuló körön amennyit lehet
  legyen a kilépő él Z_{i+1}-beli végpontja z
  $i++$
  if $d_i(z) < d_{i+1}(z)$ **then**
    Cimkézés
  end if
end while

Az implementáció helyességéhez csak annyit kell belátnunk, hogy ha a q_i él Z_{i+1}-beli végpontjának nem nőtt a cimkéje, akkor egyik csúcsnak sem nőtt a cimkéje, így nincs szükség a cimkék újraszámolására.

Nézzük végig, hogy mi változik az $i.$ és az $i+1.$ cimkék meghatározásánál! A $T_i \backslash q_i$ éleket mindkét esetben mindkét irányban használhatjuk. A $\overline{T_i \cup p_i}$ éleken a folyamérték nem változott, így mindkét esetben csak ugyanabban az irányban használhatók. p_i-t az $i.$ cimkék meghatározásánál csak az egyik irányban használhattuk, míg az $i+1.$ cimkék meghatározásánál már mindkét irányban. Végül a q_i élt az $i.$ cimkékhez mindkét irányba, az $i+1.$ cimkéknél viszont csak az egyik irányba használhatjuk. (Ha $p_i = q_i$, akkor őt az $i+1.$ cimkéknél pont az ellentétes irányban használhatjuk).

Ha egy z' csúcs cimkéje megváltozott, akkor a 3.1.3 lemma szerint csak nőhetett. Az előbbiek szerint ez csak úgy történhet, ha a legrövidebb $s \to z'$ pszeudó-javítóút x^i mellett használta a q_i élet, viszont x^{i+1} mellett már nincs $d_i(z')$ hosszú $s \to z'$ pszeudó-javítóút.

Ez az $s \to z'$ pszeudó-javítóút viszont akkor használta z-t, és oda sincs már x^{i+1} mellett $d_i(z)$ hosszú pszeudó-javítóút, vagyis ekkor z-nek is nőtt a cimkéje.

Annak eldöntéséhez, hogy z-nek nőtt-e a cimkéje elég végignéznünk z szomszédait. Ha q_i másik végpontján kívül van még olyan szomszédja, melynek $i.$ cimkéje szintén $d_i(z) - 1$, akkor x^{i+1} mellett is van $d_i(z)$ hosszú pszeudó-javítóút z-be, így a cimke nem nőtt.

3.1.4. Tétel. *A fenti implementáció $\mathcal{O}(n^2m)$ műveletet igényel.*

Bizonyítás: A cimkézést csak az algoritmus elején végezzük el, illetve akkor, ha az utolsó pivotálás során kilépő él megfelelő végpontjának nőtt a cimkéje. Mivel minden cimke egy 0 és n közti egész, és egyébként a cimkék nem csökkenhetnek, így ezt a lépést összesen $\mathcal{O}(n^2)$-szer végezhetjük el. A cimkézés egyszeri elvégzése $\mathcal{O}(m)$ művelet, így összesen $\mathcal{O}(n^2m)$ időt töltünk cimkézéssel.

A pivotálást a 3.1.2 tétel szerint $\mathcal{O}(nm)$-szer végezzük el. A minimális cimkéjű C_i-beli él megkereséséhez elég egy minimális cimkéjű Z_i-beli csúcsot keresnünk, majd az ő szomszédait végignézni, így $\mathcal{O}(n)$ művelettel találunk belépő élt. A kör, amin javítunk szintén $\mathcal{O}(n)$ hosszú. Annak eldöntéséhez, hogy z cimkéje nőtt-e, a fentiek szerint elég z szomszédait végignéznünk, ez is $\mathcal{O}(n)$ művelet. Így ezeket a lépéseket mind $\mathcal{O}(n^2m)$ idő alatt végezzük el.

Evvel a bizonyítást befejeztük.

□

3.2. Megengedett bázisfolyam előállítása az MBU megengedettségi algoritmussal

Alkalmazások során gyakran előfordul, hogy nem nulla alsó korlátos hálózatban kell maximális folyamot keresnünk. Tekintsük tehát a már megszokott lineáris programozási feladatot:

$$\begin{aligned} & \max x_{t,s} \\ \forall v \in V: \quad & \sum_{(w,v)\in E^*} x_{w,v} - \sum_{(v,w)\in E^*} x_{v,w} = 0 \\ \forall e \in E: \quad & l_e \leq x_e \leq u_e \end{aligned} \tag{3.2}$$

ahol ismét egy (t,s) él bevezetésével cirkulációs feladattá alakítottuk a hálózati folyamot.

A célunk primál megengedett bázismegoldás keresése. Ehhez egy tetszőleges bázismegoldásból indulunk ki, kiválasztunk egy nem megengedett élt, és megengedetté tesszük. Eközben ügyelünk, hogy megengedett él ne váljon nem megengedetté. Ha az algoritmus elakad, akkor a feladatnak nincs megengedett megoldása.

Az algoritmusunk vázlata a következő:

Algoritmus 4 Megengedett megoldás keresése az MBU algoritmussal

T_1 tetszőleges feszítőfa, x^1 egy T_1-hez tartozó bázismegoldás, $i = 1$.
while létezik nem megengedett él **do**
  Legyen (g, h) a legkisebb infízibilitású nem megengedett él.
  Cimkézés (g, h)-hoz.
  while (g, h) nem megengedett **do**
    if $d_i(h) = \infty$ **then**
      return Nincs megengedett megoldás!
    end if
    keressünk egy minimális cimkéjű $C_i^{g,h}$-beli élt
    javítsunk a kialakuló körön
    legyen a kilépő él Z_{i+1}-beli végpontja z
    $i++$
    if $d_i(z) < d_{i+1}(z)$ **then**
      Cimkézés (g, h)-hoz.
    end if
  end while
end while

Egy induló bázismegoldást a következőképpen kaphatunk. Vegyünk egy tetszőleges (t, s)-et tartalmazó T_1 feszítőfát. A T_1-en kívüli éleken legyen tetszőlegesen $x_e^1 = l_e$ vagy $x_e^1 = u_e$. A bázisbeli élek értékét a csúcsokra vonatkozó megmaradási egyenletek egyértelműen meghatározzák, hiszen a nekik megfelelő oszlopok lineárisan független rendszert alkotnak. Ez szemléletesen is igaz: a feszítőfa egy levelét véve a csúcsra vonatkozó egyenletben csak a hozzá tartozó bázisbeli él értéke ismeretlen, így az kiszámolható. Elhagyva ezt a csúcsot és élt a feszítőfából hasonló módon számolhatjuk ki egyenként a többi él értékét is.

Az így kapott bázismegoldás általában nem primál megengedett, ami azt jelenti, hogy néhány bázisbeli élen nem teljesülnek az alsó- vagy felső korlátok. Mivel a bázis $n-1$ elemű, így kezdetben legfeljebb $n-1$ primál nem megengedett változónk lehet. Az algoritmus során egyenként megengedetté tesszük ezeket a változókat, ezután a primál megengedett megoldásból például a Goldfarb-Hao algoritmussal optimális megoldást készíthetünk.

Egy primál nem megengedett él infízibilitásán $x_e < l_e$ esetén az $l_e - x_e$ értéket, míg $x_e > u_e$ esetén az $x_e - u_e$ értéket értjük. Legyen a legkisebb infízibilitású nem megengedett él (g, h), és tegyük fel, hogy a felső korlátját lépi át. A másik eset hasonlóan tárgyalható, csak fel kell cserélni g és h szerepét.

Az i. pivotálás előtti bázist jelölje most is T_i, a folyamot pedig x^i. A (g,h) élt elhagyva T_i-ből az két részfára esik, legyen T_i^g a g-t tartalmazó részfa, és T_i^h a h-t tartalmazó részfa.

A Goldfarb-Hao algoritmusnál elmondottak kisebb változtatásokkal átvihető erre az esetre. Mivel minden lépésben a (g,h) él infízibilitását szeretnénk csökkenteni, így az i. pivotálás során olyan belépő élt szeretnénk választani, hogy a kialakuló kör tartalmazza (g,h)-t, és a belépő él a megfelelő irányban legyen változtatható:

$$\begin{aligned} C_i := \{(v,w) \in E^* \ : \quad & x^i_{v,w} = l_{v,w}, v \in T_i^g, w \in T_i^h \\ \text{vagy} \quad & x^i_{v,w} = u_{v,w}, v \in T_i^h, w \in T_i^g\} \end{aligned}$$

Ha $C_i = \emptyset$, akkor a feladatnak nem létezik megengedett megoldása. Adjuk össze ugyanis T_i^g csúcsaira a megmaradási egyenleteket, így azt kapjuk, hogy a T_i^g-ből kimenő folyamérték megegyezik a T_i^g-be bemenő folyamértékkel. Ezen élek közül egyedül (g,h) bázisbeli, rá $x_{g,h} > u_{g,h}$, a többi kimenő élre viszont $C_i = \emptyset$ miatt $x_e = u_e$ és a bejövő élekre szintén $C_i = \emptyset$ miatt $x_e = l_e$. Így a következő egyenlőtlenséget kapjuk:

$$\sum_{v \in T_i^g} u_{v,w} < \sum_{v \in T_i^g} x_{v,w} = \sum_{v \in T_i^g} x_{w,v} = \sum_{v \in T_i^g} l_{w,v} \tag{3.3}$$

Vagyis T_i^g-ből több folyamnak kell távoznia, mint amennyi oda a bejövő éleken keresztül egyáltalán bevihető, ami szemléletesen is kielégíthetetlen feltétel. Így az algoritmus helyességének és végességének belátásával a következő tételre nyerünk konstruktív bizonyítást:

3.2.1. Tétel. *A* (3.2) *feladatnak pontosan akkor létezik megoldása, ha nincs a csúcsoknak olyan részhalmaza, melyből kimenő élek alsó korlátainak összege nagyobb, mint a bejövő élek felső korlátainak összege.*

Ez a tétel a Farkas lemma speciális esete hálózati folyamokra, megkapható a Ford–Fulkerson tétel 2.4. részben leírt kibővített gráfra való alkalmazásával is, illetve ebből a tételből is levezethető a Ford–Fulkerson tétel. További hasonlóságuk, hogy míg a maximális folyam–minimális vágás tétel egy folyam maximalitására ad könnyen ellenőrizhető objektumot, addig a 3.2.1 tétel annak ellenőrzésére ad egyszerű feltételt, hogy nincs megoldás. Valóban, ha az algoritmus leállásakor azt mondja, hogy nem talált megengedett megoldást, és megadja a T_i^g részfa csúcshalmazát, akkor pillanatok alatt leellenőrizhető, hogy az algoritmus helyes következtetésre jutott.

Ha $C_i \neq \emptyset$, akkor a Goldfarb-Hao algoritmuséhoz hasonló cimkézési technika segítségével kiválasztunk egy $(v,w) \in C_i$ belépő élet. Ekkor $T_i \cup (v,w)$-ben kialakul egy kör, melynek eleme (g,h) is. A kört a (g,h) él irányításával ellentétesen tekintve az előreéleken egy f mennyiséget növelünk, hátraélein pedig egy f mennyiséget csökkentünk, ahol f a maximális mennyiség, amivel elvégezve ezt a javítást nem válik primál megengedett él nem megengedetté, illetve ha ez több, mint (g,h) infízibilitása, akkor csak annyi:

$$f := \min\{ \begin{array}{lcl} u_e - x_e^i & : & e \text{ megengedett előreél,} \\ x_e^i - l_e & : & e \text{ megengedett hátraél,} \\ (g,h) & & \text{infízibilitása } \} \end{array}$$

A kilépő él legyen (g,h), ha az megengedetté vált (f a (g,h) infízibilitása), egyébként pedig a kör egyik éle, melyen a minimum felvétetett. Megjegyezzük, hogy a primál nem megengedett élek infízibilitása így tovább romolhat, de új infízibilis él nem alakul ki.

Egy $z \in V$ csúcs cimkéje a legrövidebb $g \to z$ pszeudó-javítóút hossza, amelyik nem használja a (g,h) élt (de használhatja a (t,s) élt):

Algoritmus 5 Cimkézés (g,h)-hoz

$d_i(g) = 0$, lista $= \{g\}$
while lista $\neq \emptyset$ **do**
  v legyen a lista első eleme
  for all $(v,w) \in E^* \backslash \{(g,h)\}$, ($(v,w) \in T_i$ vagy $x_{v,w} = l_{v,w}$), w cimkézetlen **do**
    $d_i(w) = d_i(v) + 1$
    w a lista végére kerül
  end for
  for all $(w,v) \in E^* \backslash \{(g,h)\}$, ($(w,v) \in T_i$ vagy $x_{w,v} = u_{w,v}$), w cimkézetlen **do**
    $d_i(w) = d_i(v) + 1$
    w a lista végére kerül
  end for
  töröljük v-t a listáról
end while

A (v,w) bázison kívüli él cimkéje legyen

$$\delta_i(v,w) = \begin{cases} d_i(v) & \text{ha } x_{v,w} = l_{v,w} \\ d_i(w) & \text{ha } x_{v,w} = u_{v,w} \end{cases}$$

Az e bázisbeli élre és $i \geq 2$-re legyen $\delta_i(e) = \delta_{i-1}(e)$, illetve minden $e \in T_1$-re $\delta_1(e) = 0$.

Az eddig leírtakból már összerakható az általunk javasolt algoritmus helyessége, valamint lépésszámbecslése:

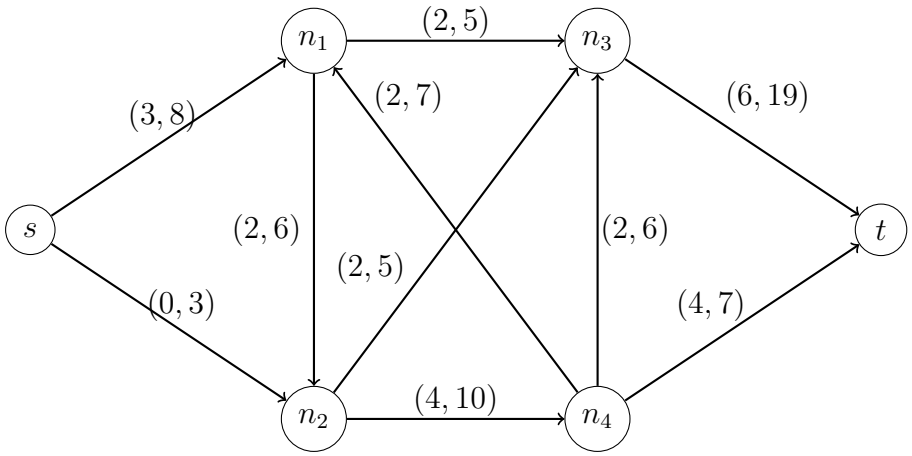
3.2.2. Tétel. *A 4. algoritmus $\mathcal{O}(n^3m)$ művelettel megtalálja a (3.2) feladat egy megengedett megoldását, ha az létezik, vagy kimutatja, hogy a feladat nem megoldható.*

Bizonyítás: az algoritmus belső ciklusa $\mathcal{O}(n^2m)$ művelettel kijavítja a (g, h) élt, ennek bizonyítása szinte szóról szóra megegyezik a Goldfarb-Hao algoritmusnál leírtakkal. Mivel kezdetben legfeljebb $n-1$ nem megengedett élünk van, és a belső ciklus minden lefutása után legalább 1 élt megengedetté tettünk, miközben a már megengedett éleket nem rontottuk el, így $\mathcal{O}(n)$-szer végezzük el a belső ciklust. Így összesen az algoritmusunk $\mathcal{O}(n^3m)$ műveletet végez.

□

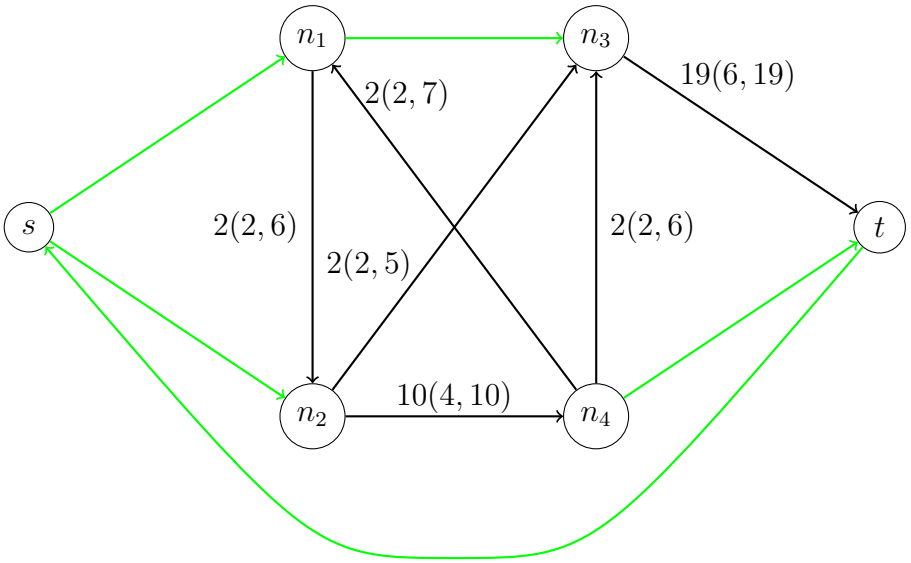
3.3. Példa

Tekintsük a következő hálózatot, és a hozzá tartozó lineáris programozási feladatot:



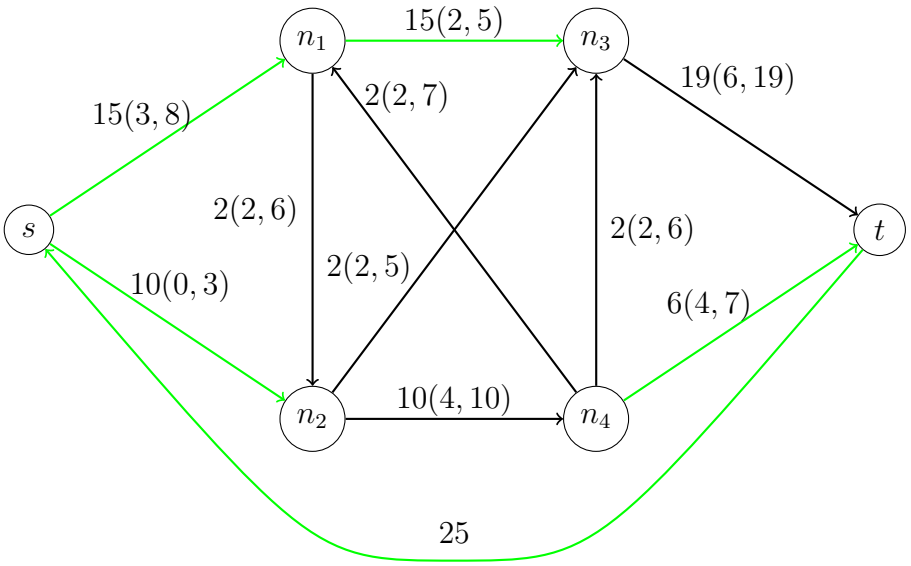
$$
\begin{aligned}
\max \quad & x_{t,s} \\
\forall e \in E \quad : \quad & l_e \leq x_e \leq u_e \\
-x_{s,1} - x_{s,2} + x_{t,s} &= 0 \\
x_{s,1} - x_{1,2} - x_{1,3} + x_{4,1} &= 0 \\
x_{s,2} + x_{1,2} - x_{2,3} - x_{2,4} &= 0 \\
x_{1,3} + x_{2,3} + x_{4,3} - x_{3,t} &= 0 \\
-x_{4,1} + x_{2,4} - x_{4,3} - x_{4,t} &= 0
\end{aligned}
$$

A (t, s) él behúzásával alakítsuk a feladatot cirkulációs problémává. Vegyünk egy tetszőleges feszítőfát, mondjuk az $\{(s, n_1), (s, n_2), (n_1, n_3), (n_4, t), (t, s)\}$ éleket. Ezután a bázison kívüli élek értékét állítsuk valamelyik korlátjukra:



3.1. ábra. Feszítőfa, és bázison kívüli élek

A csúcsokra vonatkozó megmaradási egyenletekből kiszámolhatók a bázisbeli élek értékei. Például n_3 egyenletéből megkapjuk az (n_1, n_3) él értékét, ezután n_1 egyenletéből az (s, n_1) él értékét... satöbbi.



3.2. ábra. Kiinduló bázismegoldás

A szimplex tábla felírásához x_e helyett vezessük be a következő változókat: $s_e = x_e - l_e$ és $z_e = u_e - x_e$. Ekkor a változóink nemnegatívak, és felső korlát helyett az $s_e + z_e = u_e - l_e$ egyenleteket teljesítik. Ha $e \in T_i$, akkor az s_e és a z_e változó is bázisbeli, $e \notin T_i$ és $x_e = l_e$ esetén csak z_e bázisbeli, illetve $x_e = u_e$ esetén csak s_e.

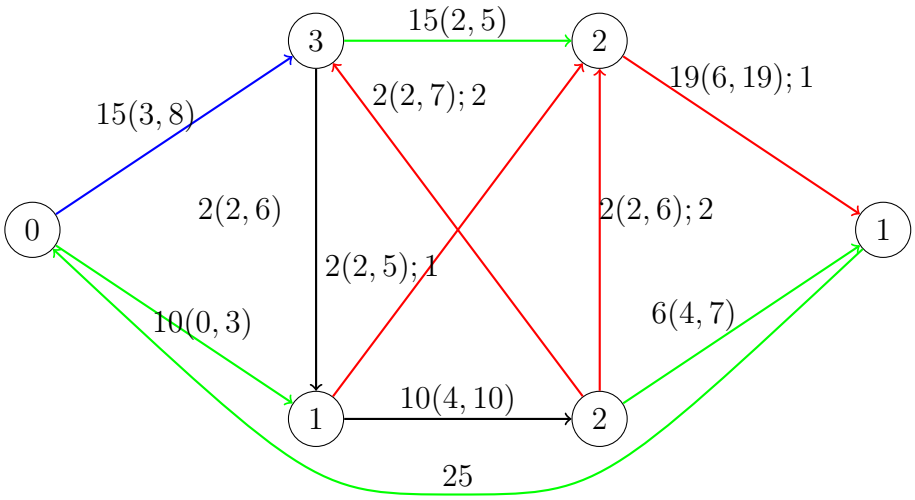
	s_3	s_5	s_6	s_8	z_7	z_9	
s_{11}		1		1	1	1	25
s_1	-1	1	1	1		1	12
s_2	1		-1		1		10
s_4			1	1		1	13
s_{10}		1		1	1		2
z_1	1	-1	-1	-1		-1	-7
z_2	-1		1		-1		-7
z_3	1						4
z_4			-1	-1		-1	-10
z_5		1					5
z_6			1				3
s_7					1		6
z_8				1			4
s_9						1	13
z_{10}		-1		-1	-1		1
		1		1	1	1	

3.3. ábra. Induló pivot tábla

A gráfból leolvasható, hogy az (s, n_1), (s, n_2) és (n_1, n_3) élek nem teljesítik a felső korlátjukat. Az infízibilitásaik rendre 7, 7, és 10. Ugyanezt az információt a szimplex táblában a z_1, z_2 és z_4 változók negatív értékeiből látjuk, melyek abszolút értékei szintén az infízibilitás mértékét adják. Megjegyezzük, hogy a szimplex tábla célfüggvénysorából láthatjuk, hogy duál megengedett megoldásból indulunk, hiszen mindegyik érték nemnegatív. Ez azért van, mert az $S \to Z$ éleket a felső korlátjukra, míg a $Z \to S$ éleket az alsó korlátjukra állítottuk. Most ugyan nem célunk a folyam maximalizálása, így a célfüggvénysor feltüntetése feleslegesnek tűnhet, de majd láthatjuk, amikor a bázis duál megengedettsége elveszik.

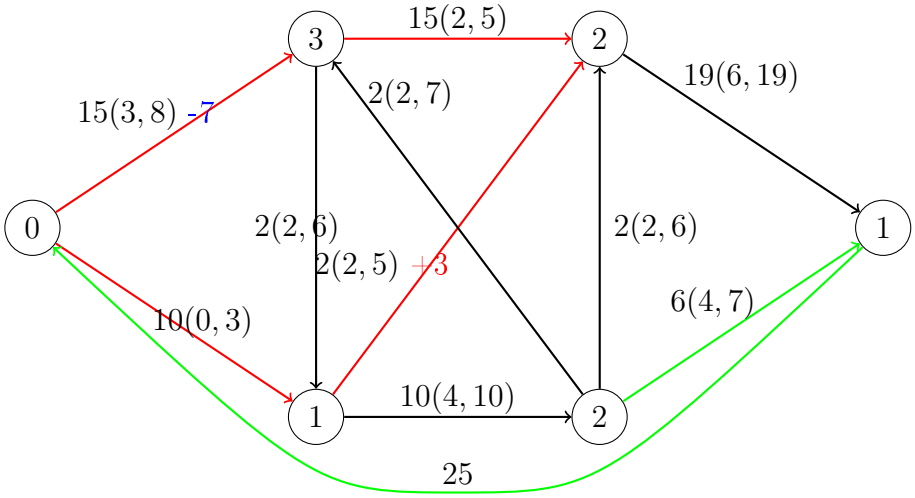
A két legkisebb infízibilitású nem megengedett él közül válasszuk mondjuk az (s, n_1) élt. Ekkor a T_1 feszítőfa (s, n_1) elhagyásával két részre esik: T_1^s tartalmazza az s, n_2, n_4 és t csúcsokat, és $T_1^{n_1}$ tartalmazza n_1-et és n_2-t.

Végezzük el a cimkézést s-ből (3.4 ábra).



3.4. ábra. Cimkézés

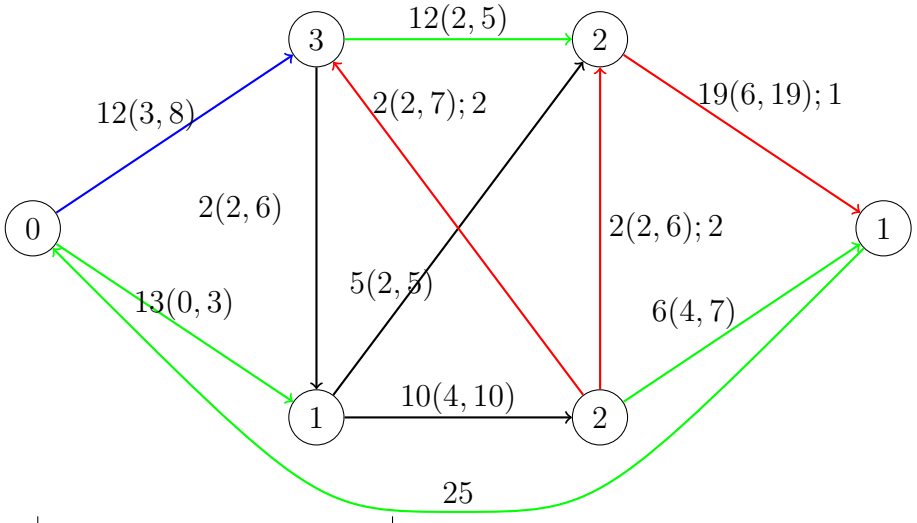
A lehetséges belépő élek $C_1 = \{(n_2, n_3), (n_4, n_1), (n_4, n_3), (n_3, t)\}$. Vegyük észre, hogy ezek az élek pont a z_1 sorában szereplő negatív elemeknek felelnek meg. Ezek közül (n_2, n_3) és (t, n_3) cimkéje minimális, válasszuk mondjuk (n_2, n_3)-at.



3.5. ábra. Javítás a körön

A feszítőfához hozzávéve az (n_2, n_3) élt kialakul az $s \to n_2 \to n_3 \to n_1$ kör. (s, n_2)-n 7-et kell javítanunk, hogy megengedetté váljon, (n_2, n_3) nem megengedett, így a javítás szempontjából nem számít, (n_2, n_3)-on 3-at tudunk növelni a primál megengedettség megsértése nélkül, és végül (s, n_2) szintén nem megengedett, így ő sem számít. Összességében tehát 3-at tudunk javítani, és (n_2, n_3) távozik a bázisból.

Nézzük meg az előző számolást a szimplex táblán (3.3 ábra). A z_1 vezérváltozó sorából az s_6 változóhoz tartozó -1-est választottuk. Ezután elvégeztünk egy primál hányadostesztet az s_6 oszlop azon pozitív elemein, melyekhez tartozó jobboldal nemnegatív (a nem megengedett z_2 és z_4 változókat nem vettük figyelembe), illetve a vezérváltozón. A legkisebb hányadost z_6 adta, így s_6 lép be a bázisba, és z_6 távozik.



	s_3	s_5	s_8	z_6	z_7	z_9	
s_{11}		1	1		1	1	25
s_1	-1	1	1	-1		1	9
s_2	1			1	1		13
s_4			1	-1		1	10
s_{10}		1	1		1		2
z_1	1	-1	-1	1		-1	-4
z_2	-1			-1	-1		-10
z_3	1						4
z_4			-1	1		-1	-7
z_5		1					5
s_6				1			3
s_7					1		6
z_8			1				4
s_9						1	13
z_{10}		-1	-1		-1		1
		1	1		1	1	

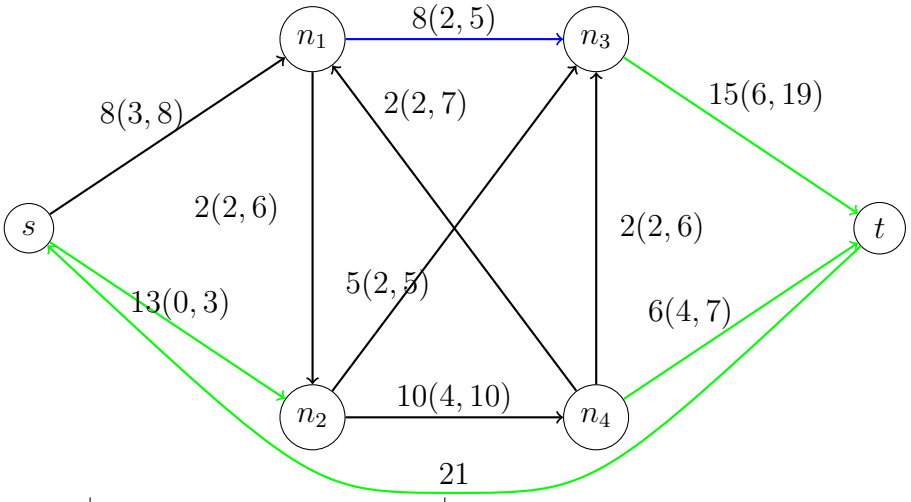
$T_2^s = \{s, n_2, n_4, t\}$
$T_2^{n_1} = \{n_1, n_3\}$
$C_2 = \{(n_4, n_1), (n_4, n_3), (n_3, t)\}$
(5-ös, 8-as és 9-es él)

3.6. ábra. Az első pivotálás után

A kilépő él $T_2^{n_1}$-beli végpontja n_3. Az ő cimkéje nem változik, mivel van még 1-gyel kisebb cimkéjű szomszédja (a t csúcs), így a csúcsok cimkéi nem változtak.

Mivel ugyanaz az él lépett be az első pivotálásnál, mint amelyik kilépett, a feszítőfa változatlan maradt. A lehetséges belépő élek $C_2 = \{(n_4, n_1), (n_4, n_3), (t, n_3)\}$, közülük a legkisebb cimkéjű él (t, n_3).

A kialakuló $s \to t \to n_3 \to n_1$ körön 4-et javítunk, így (s, n_1) lecsökken a felső korlátjára és megengedetté válik.



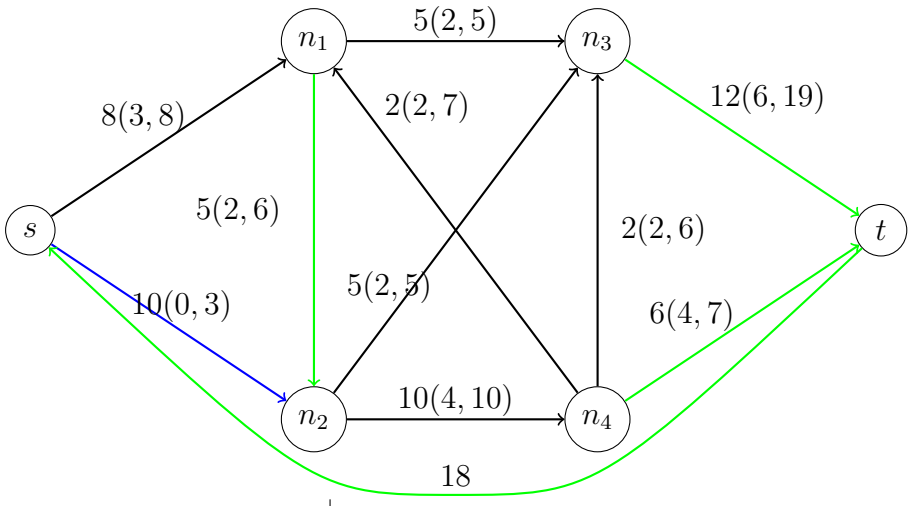
	s_3	s_5	s_8	z_1	z_6	z_7	
s_{11}	1			1	1	1	21
s_1				1			5
s_2	1				1	1	13
s_4	1	-1		1			6
s_{10}		1	1			1	2
z_9	-1	1	1	-1	-1		4
z_2	-1				-1	-1	-10
z_3	1						4
z_4	-1	1		-1			-3
z_5		1					5
s_6					1		3
s_7						1	6
z_8			1				4
s_9	1	-1	-1	1			9
z_{10}		-1	-1			-1	1
	1			1	1	1	

$T_3^{n_1} = \{n_1\}$
$T_3^{n_3} = \{s, n_2, n_3, n_4, t\}$
$C_3 = \{(s, n_1), (n_1, n_2)\}$
(1-es és 3-as él)

3.7. ábra. A második pivotálás után

Az (s, n_1) él megengedetté vált. A maradék két nem megengedett él ((s, n_2) és (n_1, n_3)) közül (n_1, n_3) infizibilitása kisebb, így őt tesszük először megengedetté. Vegyük észre, hogy az (s, n_2) él infizibilitása nőtt az első él megengedetté tétele közben.

Elvégezzük a cimkézést n_1-ből n_3-ba. A lehetséges belépő élek (s, n_1) és (n_1, n_2). A két él cimkéje azonos, így válasszuk mondjuk (n_1, n_2)-t. A kialakuló körön 3-at javítva az (n_1, n_3) él megengedetté válik, és kilép a bázisból.

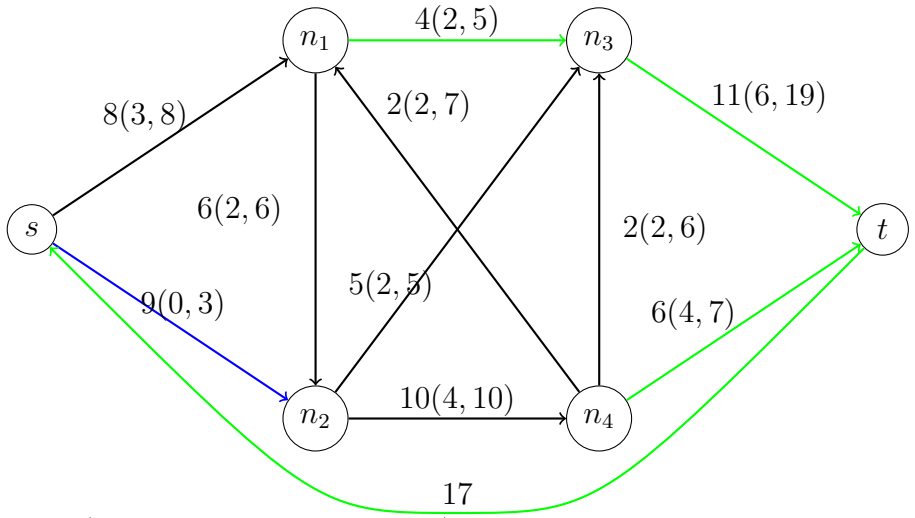


	s_5	s_8	z_1	z_4	z_6	z_7	
s_{11}	1			1	1	1	18
s_1			1				5
s_2	1		-1	1	1	1	10
s_4				1			3
s_{10}	1	1				1	2
z_9		1		-1	-1		7
z_2	-1		1	-1	-1	-1	-7
z_3	1		-1	1			1
s_3	-1		1	-1			3
z_5	1						5
s_6					1		3
s_7						1	6
z_8		1					4
s_9		-1		1			6
z_{10}	-1	-1				-1	1
	1			1	1	1	

$T_4^s = \{s, n_3, n_4, t\}$
$T_4^{n_2} = \{n_1, n_2\}$
$C_4 = \{(n_1, n_3), (n_4, n_1), (n_2, n_3), (n_2, n_4)\}$
(4-es, 5-ös, 6-os és 7-es él)

3.8. ábra. A harmadik pivotálás után

Végül az (s, n_2) élt szeretnénk megengedetté tenni. Elvégezzük s-ből n_2-be a cimkézést. A lehetséges belépő élek az (n_1, n_3), (n_2, n_3), (n_4, n_1) és (n_2, n_4). Mindegyik él cimkéje 2, válasszuk mondjuk (n_1, n_3)-at. A kialakuló körön 1-t tudunk javítani, (n_1, n_2) távozik a bázisból.



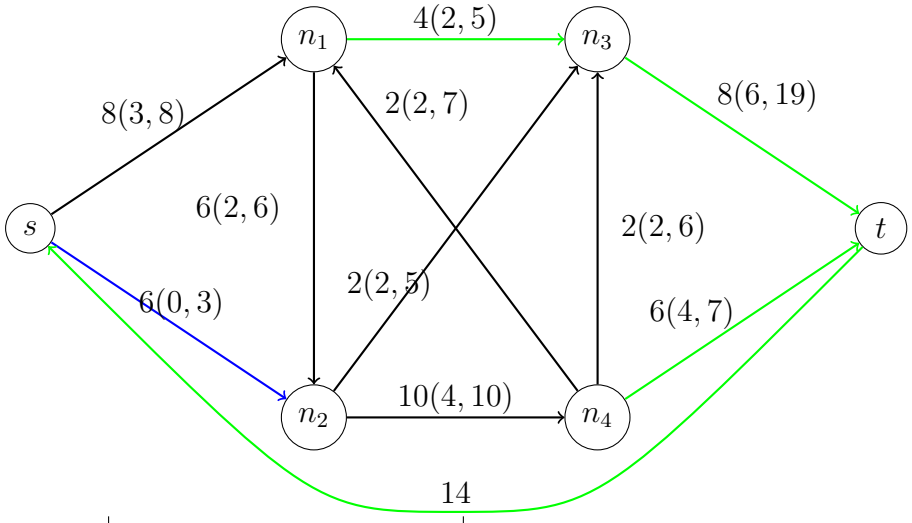
	s_5	s_8	z_1	z_3	z_6	z_7	
s_{11}			1	-1	1	1	17
s_1			1				5
s_2				-1	1	1	9
s_4	-1		1	-1			2
s_{10}	1	1				1	2
z_9	1	1	-1	1	-1		8
z_2				1	-1	-1	-6
z_4	1		-1	1			1
s_3				1			4
z_5	1						5
s_6					1		3
s_7						1	6
z_8		1					4
s_9	-1	-1	1	-1			5
z_{10}	-1	-1				-1	1
			1	-1	1	1	

$T_5^s = \{s, n_1, n_3, n_4, t\}$
$T_5^{n_1} = \{n_2\}$
$C_5 = \{(n_2, n_3), (n_2, n_4)\}$
(6-os és 7-es él)

3.9. ábra. A negyedik pivotálás után

Figyeljük meg, hogy a bázismegoldás duál megengedettsége elveszett az előző pivotálás során.

A lehetséges belépő élek (n_2, n_3) és (n_2, n_4). Mindkét él cimkéje 2, így válasszuk mondjuk az (n_2, n_3)-at. A kialakuló körön 3-at tudunk javítani, (n_2, n_3) kilép a bázisból.



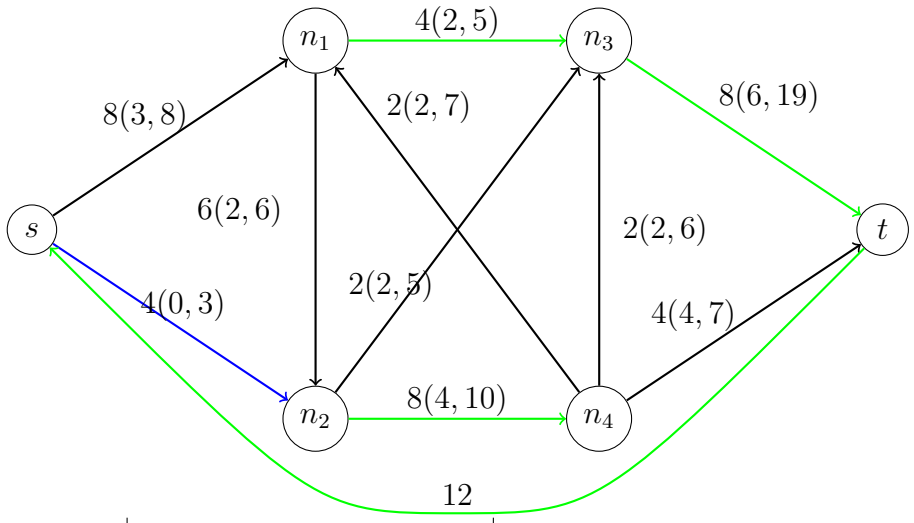
	s_5	s_6	s_8	z_1	z_3	z_7	
s_{11}		-1		1	-1	1	14
s_1				1			5
s_2		-1			-1	1	6
s_4	-1			1	-1		2
s_{10}	1		1			1	2
z_9	1	1	1	-1	1		11
z_2		1			1	-1	-3
z_4	1			-1	1		1
s_3					1		4
z_5	1						5
z_6		1					3
s_7						1	6
z_8			1				4
s_9	-1		-1	1	-1		5
z_{10}	-1		-1			-1	1
		-1		1	-1	1	

$T_6^s = \{s, n_1, n_3, n_4, t\}$
$T_6^{n_2} = \{n_2\}$
$C_6 = \{(n_2, n_4)\}$
(7-es él)

3.10. ábra. Az ötödik pivotálás után

Az egyetlen lehetséges belépő él az (n_2, n_4), a kialakuló körön 2-t tudunk javítani, az (n_4, t) él lép ki a bázisból.

Ekkor a következőképpen néz ki a hálózat:



	s_5	s_6	s_8	s_{10}	z_1	z_3	
s_{11}	-1	-1	-1	-1	1	-1	12
s_1					1		5
s_2	-1	-1	-1	-1		-1	4
s_4	-1				1	-1	2
z_7	1		1	1			2
z_9	1	1	1		-1	1	11
z_2	1	1	1	1		1	-1
z_4	1				-1	1	1
s_3						1	4
z_5	1						5
z_6		1					3
s_7	-1		-1	-1			4
z_8			1				4
s_9	-1		-1		1	-1	5
z_{10}				1			3
	-1	-1	-1	-1	1	-1	

$T_7^s = \{s, n_1, n_3, t\}$

$T_7^{n_2} = \{n_2, n_4\}$

$C_7 = \emptyset$

3.11. ábra. Végső állapot

Az (s, n_2) él még nem vált megengedetté, de több pivotálást már nem tudunk végrehajtani. A feladatnak tehát nincs megengedett megoldása. Ez könnyen ellenőrizhető, ha tekintjük a végső bázis egyik részfájának csúcsait, mondjuk $\{n_2, n_4\}$-et.

A bejövő élek felső korlátainak összege $3+6=9$, míg a kimenő élek alsó korlátainak összege $2+2+2+4=10$, tényleg nem létezhet megengedett megoldás.

A szimplex táblán szintén látszik, hogy nem végezhető el több pivotálás, hiszen a vezérváltozó értéke még mindig negatív, de a sorában már nem található több negatív elem.

4. fejezet

Alkalmazás: mozdony-hozzárendelési probléma

Az ipari forradalom óta a vasút a nagyobb volumenű személy- és teherszállítás egyik legolcsóbb módja, csak a vizi szállítással lehetne összehasonlítani, ami erősen függ az adott terület földrajzi viszonyaitól. Így az egyes vasúttársaságok egyre növekvő igényekkel néztek szembe. Egy idő után az erőforrások hatékony allokációja már nem triviális feladat, mint ahogy az így megtakarított pénzösszeg sem. Eleinte ez a közlekedésmérnökök empírikus módszerei alapján történt, ami persze nem feltétlenül optimális.

A vasúttársaságok feladatainak mai mérete mellett indokolt komplexebb matematikai modellek felállítása és elemzése. Gyönyörű példa a holland vasúttársaság sikertörténete [49]. A társaság szinte összes tevékenyége matematikai modellek mentén lett átszervezve, például a menetrend, a szerelvények összeszerelése, a személyzet hozzárendelés. A vállalat becsült haszna évi 70 millió euróra rúg, és javult a szolgáltatás pontossága.

Mi ebben a szakdolgozatban a mozdony-hozzárendelési problémán fogjuk illusztrálni a bemutatott algoritmusokat. Először felépítjük a probléma matematikai modelljét, illetve annak különböző változatait. Ezután vázoljuk az algoritmusok implementációt és ismertetjük a futási eredményeinket.

4.1. A modell

Adott egy régió menetrendje egy bizonyos időszakra, például egy napra vagy egy hétre. A menetrend a következő adatokat tartalmazza a vonatokról: indulási és érkezési idő, indulási és érkezési hely, a vontatáshoz használható minimális és maxi-

mális mozdonyszám.

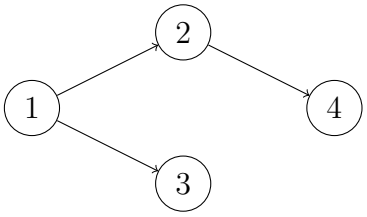
Egy mozdony másik vonatra való átszerelése egy adott τ ideig tart, ezt "technikai időnek" nevezzük. Ez az idő függhet a szerelvény és a mozdony típusától, illetve az adott állomás felszereltségétől is, de mivel ilyen részletes adatok nem álltak rendelkezésünkre, a modellben konstans 30 perccel számoltunk. A vasúttársaság első kérdése például az lehet, hogy a jelenlegi gépparkjukkal kivitelezhető-e a menetrend, illetve ha igen, akkor adjunk meg egy megfelelő mozdony-hozzárendelést. Ennél egy kicsit többet mondunk, ha meghatározzuk a minimális számú mozdonyt, amivel teljesíthető a menetrend, és megadunk egy minimális mozdony-hozzárendelést. Ez egyébként is kívánatos, hiszen így könnyebben pótolható egy esetleg meghibásodó mozdony, illetve az éppen nem használt mozdonyok elküldhetők karbantartásra.

A modell felépítésében a [34] és [35] dolgozatokat követjük. Az adatok alapján megkonstruáljuk az úgynevezett "vonatgráfot". A $G(V, E)$ irányított gráf csúcsai reprezentálják a vonatokat, és legyen $(i, j) \in E$ pontosan akkor, ha az i vonatról át tudunk kapcsolni mozdonyt a j vonatra, vagyis ha

$$\text{érkezési hely}(i) = \text{indulási hely}(j)$$
$$\text{érkezési idő}(i) + \tau \leq \text{indulási idő}(j)$$

Ekkor a gráf egy irányított útja megfelel egy olyan vonatsorozatnak, melyhez ugyanazt a mozdonyt tudjuk használni. Ebbe beleértjük az egy csúcsból álló "utat" is, hiszen az azt jelenti, hogy az adott mozdonnyal csak azt az egy vonatot vontatjuk. Így a feladatunk a G gráf minimális számú úttal való fedése úgy, hogy minden csúcs a megfelelő minimális és maximális mozdonyszám közötti úttal legyen fedve. Megjegyezzük, hogy a gráf nem tartalmaz irányított kört, hiszen a mozdonyaink (egyelőre) nem képesek az időutazásra.

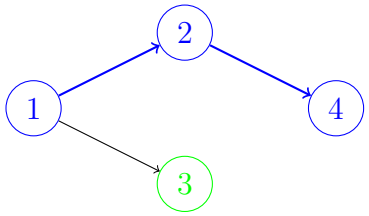
Tekintsük az alábbi egyszerű példát:



4.1. ábra.

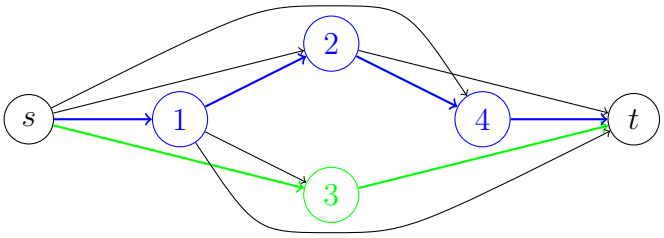
Ha minden vonathoz 1 a minimális mozdonyszám, akkor az $1, 2$ és 4 vonatokat elvontathatjuk ugyanazzal az egy mozdonnyal, és a 3-as vonatot egy külön moz-

donnyal (egy másik minimális megoldás lenne, ha a 2 és 4, valamint az 1 és 3 vonatokat vontatnánk egy-egy mozdonnyal):



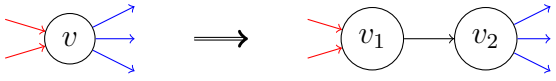
4.2. ábra.

A modellt általalakíthatjuk hálózati folyammá. Vezessünk be egy új s és t csúcsot, valamint minden $v \in V$ csúcshoz vezessünk be egy $s \to v$ és egy $v \to t$ élet. Ekkor egy eddigi $v_1 \to \ldots \to \ldots v_k$ út megfeleltethető egy 1-értékű $s \to t$ folyamnak, egyszerűen csak az elejére rakjuk s-t, és a végére t-t: $s \to v_1 \to \ldots \to v_k \to t$. Könnyen látható, hogy ez a megfeleltetés kölcsönös.



4.3. ábra.

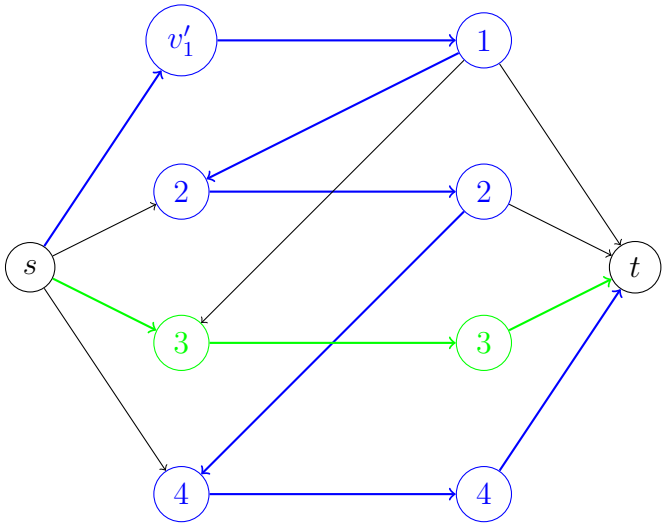
Ebben a modellben a célunk tehát minimális $s \to t$ folyamot találni, ami kielégíti minden csúcsra a megadott alsó- és felső korlátokat. Az ismertetett algoritmusok élekre vonatkozó korlátok esetén működnek, ezért tovább alakítjuk a modellünket. Egy korlátokkal rendelkező csúcsot kicserélünk két csúcsra. A bejövő éleit az első csúcsba irányítjuk, a kimenő éleit a második csúcsból vezetjük ki, továbbá összekötjük a két csúcsot és erre az új élre követeljük meg az eredeti csúcsra vonatkozó korlátokat:



4.4. ábra.

Ezt az átalakítást végezzük el a folyamon: minden vonatot jelképező i csúcs

helyett bevezetünk egy v_i' és egy v_i'' csúcsot. Ekkor a következőképpen fog kinézni a példánk:



4.5. ábra.

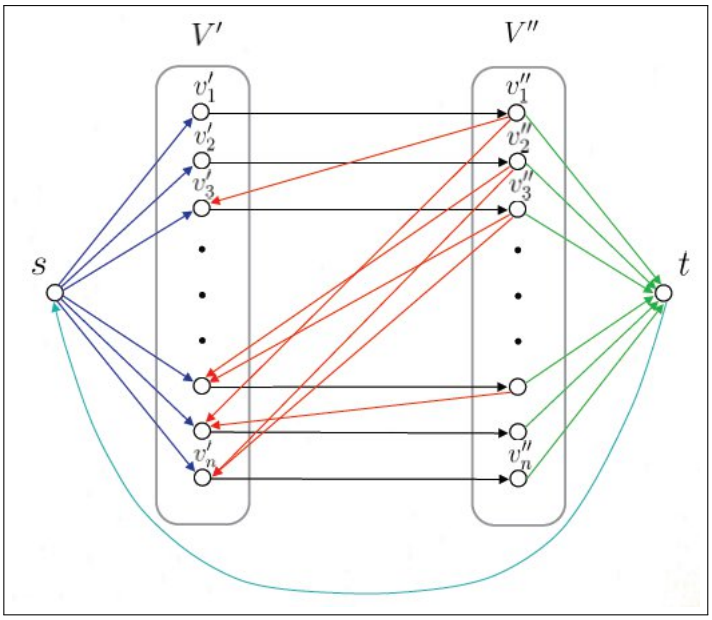
Ekkor az élekre vonatkozó alsó és felső korlátok a következőképpen néznek ki:

$$\begin{aligned} e = (v_i', v_i'') \Rightarrow & l(e) = \text{Az } i. \text{ vonat minimális mozdonyainak száma} \\ & u(e) = \text{Az } i. \text{ vonat maximális mozdonyainak száma} \\ e \neq (v_i', v_i'') \Rightarrow & l(e) = 0 \\ & u(e) = \infty \end{aligned}$$

Általánosan a feladat struktúrája a 4.6 ábrán látható (itt a (t, s) él behúzásával már cirkulációs feladattá alakítottuk át a problémát).

A felépített modellbe könnyedén beépíthetők más, a gyakorlat során előjövő feltételek. Ilyen például az úgynevezett gépmenetek alkalmazása. Gépmenetnek nevezzük, amikor egy mozdony hozzákapcsolt szerelvény nélkül, "üresen" megy át az egyik állomásról a másikra. Nyilván gépmenetek használatával tovább csökkenthető a használt mozdonyok száma, bár ez egyben felesleges költségeket is jelent, ha a menetrendet egyébként is teljesíteni lehet. További probléma, hogy a gépmenetek hozzávétele a későbbi személyzet-hozzárendelést is bonyolultabbá teszi.

A gépmeneteket a következőképpen lehet hozzávenni a modellhez. Legyen $\tau_{a,b}$ az az időtartam, ami ahhoz kell, hogy egy mozdonyt az a állomáson lekapcsoljunk egy szerelvényről, a j állomásig elmenjen gépmenetben, majd egy a b állomáson levő szerelvényhez csatlakoztassunk. Ekkor egy mozdony el tudja vontatni az a állomásra



4.6. ábra. Mozdony-hozzárendelési probléma

érkező i szerelvény után a b állomásról induló j szerelvényt, ha

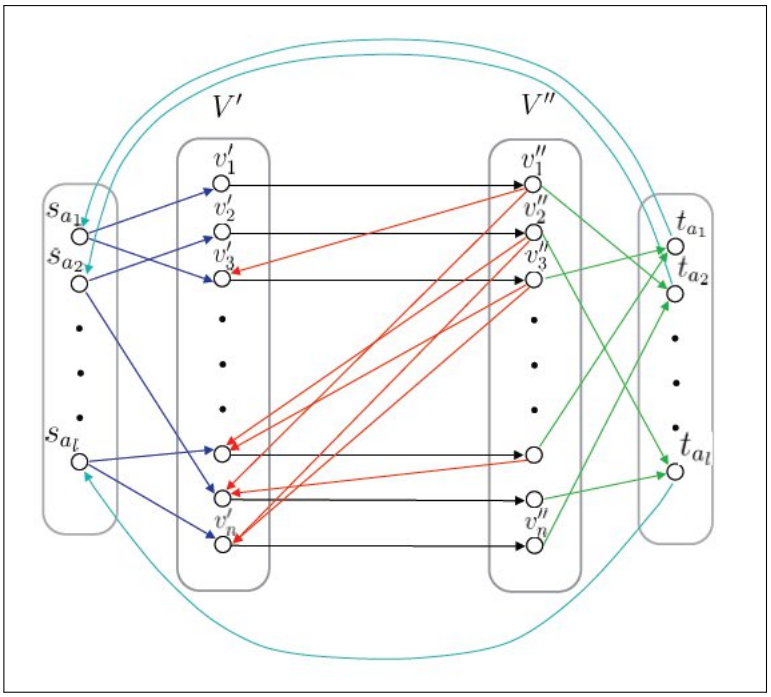
$$\text{érkezési idő}(i) + \tau_{a,b} \leq \text{ indulási idő } (j)$$

Ebben az esetben természetesen nem kell külön megszabnunk, hogy i érkezési helye megegyezzen j indulási helyével. A modellbe így elég bevennünk azon (v''_i, v'_j) éleket, amire az előző egyenlőtlenség teljesül (0 alsó és ∞ felső korláttal).

Személyszállításnál felmerülő igény, hogy a menetrend "ismételhető" legyen, hiszen a menetrend általában napi vagy heti periodicitással rendelkezik. Ez mindössze annyit jelent, hogy minden állomáson a menetrend elvégzése után ugyanannyi mozdonyt szeretnénk találni, mint amennyi az elkezdésnél volt, hiszen ekkor változatlan módon végezhető el a következő periódus is.

Ebben az esetben nem csak egy s és t csúcsot vezetünk be a vonatokat reprezentáló v' és v'' csúcsok mellé, hanem minden a állomáshoz egy külön s_a forrást, és t_a nyelőt. Ekkor $(s_a, v'_i) \in E$ pontosan akkor, ha az i vonat indulási állomása a, és $(v''_i, t_a) \in E$ pontosan akkor, ha az i vonat érkezési állomása a. Ezután cirkulációs feladattá alakítjuk a modellt a (t_a, s_a) élek behúzásával, ami biztosítja, hogy minden a állomásból ugyanannyi mozdony induljon, mint amennyi megérkezik. A mozdonyszám minimalizálásához ekkor a (t_a, s_a) élek folyamértékeinek összegét kell minimalizálnunk (4.7 ábra). Az optimalizálás már nem kezelhető Goldfarb és Hao algoritmusával, de megengedett megoldást kereshetünk az MBU megengedettségi algoritmusával.

Tehervontatás esetén a menetrend sokkal előreláthatatlanabb, viszont rövidebb



4.7. ábra. Mozdony-hozzárendelési probléma, ismételhetőséggel

a tervezés és a végrehajtás között eltelő idő. Ez azt jelenti, hogy a modellben azt is figyelembe kell vennünk, hogy az időszak elején éppen melyik állomáson mennyi mozdony található. Ekkor az ismételhetőségi feltételhez hasonlóan itt is minden állomáshoz külön s_a forrást vezessünk be, a nyelőből maradhat egy. Az aktuálisan az adott a állomáson tartózkodó mozdonyok száma felső korlátot jelent az s_a csúcson átmenő folyam értékére, ezt a már ismertetett módszerrel átalakíthatjuk élekre vonatkozó korláttá.

Végül egy fontos korlát még a gyakorlatban, hogy minden egyes mozdonynak bizonyos időközönként karbantartáson kell átesnie. Ez a feltétel megköveteli a mozdonyok egyedi kezelését, ellentétben a korábbi modellekkel, ahol csak azt tekintettük, hogy merre hány mozdony megy. Ez NP-teljes feladatot eredményez, az itt használható modellekről például [34]-ben található részletesebb leírás.

4.2. Futási eredmények

Elvégeztük a probléma implementálását a MOSEL modellezési nyelven [6]. A kód lényegében beolvassa a menetrendet, ebből generálja a lineáris programozási modell egyenleteit, illetve egyenlőtlenségeit, majd meghívja az XPRESS-MP optimalizálási programcsomag szolverét.

Másrészt elvégeztük a Goldfarb-Hao algoritmus, és a belőle képzett első fázis

algoritmus C++ nyelven való implementálását. A menetrend beolvasása után felépítettük az előző alfejezetben ismertetett gráfot, ehhez a LEMON könyvtár [20] adatstrukutúráit és rutinjait használtuk.

A mozdony-hozzárendelési probléma legegyszerűbb változatában ismert kiinduló bázismegoldás: választhatjuk az (s, v_i') és (v_i'', t) éleket bázisnak. Ez annak a hozzárendelésnek felel meg, hogy minden szerelvényt külön mozdonnyal vontatunk el. Elkészítettük a C++ implementáció olyan változatát, ami ebből a megengedett megoldásból indul ki.

A nagy méretű heti menetrendekhez írtunk továbbá egy olyan változatot, ami szekvenciálisan végzi el az optimalizálást.Ez alatt azt értjük, hogy először elvégzi az első napi menetrend optimalizálását, majd hozzáveszi a gráfhoz a második nap vonatait, és elvégzi az első két nap együttes optimalizálását, az induló megoldásnál felhasználva az első nap eredményét, és így tovább.

Tesztfeladatként felhasználtunk egy Balassagyarmat környéki napi- és heti menetrendet, egy Budapest elővárosai napi- és heti- menetrendet, valamint a Szombathelyi régió heti menetrendjét. A modellek nagyságrendjéről a következő táblázat nyújt képet (mindegyik modellt a $\tau = 30$ perc paraméterrel tekintettük):

Menetrend	vonatok	csúcsok	élek	optimális megoldás
Balassagyarmat napi	92	186	928	21
Balassagyarmat heti	644	1.290	44.416	22
Elővárosi napi	369	740	16.298	45
Elővárosi heti	2.583	5.168	855.380	47
Szombathelyi heti	6.282	12.566	2.934.775	360

A futási idők az említett algoritmusokkal:

Menetrend	XPRESS-MP	GH	GH + induló mo.	szekvenciális
Balassagyarmat napi	0.109s	0.015s	0.031s	–
Balassagyarmat heti	4.736s	8s	4.071s	0.904s
Elővárosi napi	0.2s	2.948s	0.327s	–
Elővárosi heti	2m 24s	91m 57s	9m 4s	1m 56s
Szombathelyi heti	–*	–	57m 25s	17m 14s

*A szombathelyi menetrenden az XPRESS-MP elfogyasztja az általa látott 1.8 gigabájt memóriát, majd félbeszakítja a számolást.

Látható, hogy a Goldfarb-Hao algoritmus általános hálózati folyamot megoldó implementációja elmarad az XPRESS-MP futási időitől, de ez érthető egy évtizede

fejlesztés alatt álló, piacvezető operációkutatási programcsomagnál. Ellenben a feladat speciális struktúráját kihasználó implementáció már felveszi a versenyt, és a feladat méretéhez képest méltányos idő alatt szolgáltatja az optimális megoldást egy mezei laptopon.

5. fejezet

Összefoglalás és további kutatási irányok

A dolgozatban az egytermékes hálózati folyam bizonyos algoritmikus kérdéseivel foglalkoztunk. Röviden bemutattuk a modell alkalmazási területeit és összefoglaltuk a különböző megoldási technikák történetét és hatékonyságát.

Egy vasút-optimalizálási probléma kapcsán előtérbe kerültek a probléma pivot-algoritmussal történő megoldásának kérdései. A terület egyik legjelentősebb eredménye Goldfarb és Hao nevéhez fűződik, nevezetesen az első polinomiális algoritmus kifejlesztése nulla alsó korlátú hálózati folyam feladatra. Ugyan létezik a nem nulla alsó korlátos feladatra egy hatékonynak mondható visszavezetési technika (talán ezért is keveset kutatott terület a probléma), de minket érdekelt egy direkt algoritmus kifejlesztésének lehetősége.

Ez sikerült, Goldfarb és Hao algoritmusának továbbgondolása és a polinomialitás bizonyítása képezi e szakdolgozat fő eredményét. Ez egyrészt új eredmény, mivel tudomásunk szerint ez az első polinomiális pivot algoritmus, mely átalakítások nélkül, egyenesen az eredeti feladatot oldja meg. Másrészt a részletek kidolgozása közben észrevettük, hogy tulajdonképpen az általános lineáris programozási feladat egy már ismert pivotalgoritmusa, az Anstreicher−Terlaky-féle monoton szimplex módszer [5] fízibilitási változatának [9] hálózati folyam feladatra való specializációjáról van szó. Így az eredményünket úgy is megfogalmazhatjuk, mint egy ismert pivot algoritmus hálózati folyamokon való polinomialitásának bizonyítása, a megfelelő pivotálási szabály mellett.

Természetesen adódik tehát az egyik lehetséges kutatási irány. Vajon más ismert pivot algoritmusról (például a criss-cross algoritmusról, [52], [53]) is bizonyítható-e a hálózati folyamokon való polinomialitás? Az MBU-val kapcsolatos tapasztalatok

biztatóak, a modell alapját adó gráfstruktúra elég speciálissá teszi a lineáris programozási feladatot, hogy a pozitív választ tartsuk valószínűbbnek.

A másik irányt a probléma praktikus oldala, a témát ihlető vasút-optimalizálási feladat adja. Az elméleti eredmények után bemutattuk a gyenge mozdony-hozzárendelési probléma modelljeit, és az általunk készített implementációk futásideit valós méretű menetrendeken. Az eredmények ígéretesek, de természetesen lehetne még javítani rajtuk. Még érdekesebb viszont az algoritmus többtermékes hálózati folyamra való esetleges általánosíthatósága. Ez egy hagyományosan nehezebb probléma, melyekre a klasszikus kombinatorikus algoritmusok nem nyújtanak hatékony megoldást, így különösen hasznos lenne a kérdés vizsgálata.

Irodalomjegyzék

[1] R. K. Ahuja, M. Kodialam, A. K. Mishra, and J. B. Orlin. Computational investigations of maximum flow algorithms. *European Journal of Operational Research*, 97:509–542, 1997.

[2] R. K. Ahuja and J. B. Orlin. A fast and simple algorithm for the maximum flow problem. *Operations Research*, 37:748–759, 1989.

[3] R. K. Ahuja, J. B. Orlin, and R. E. Tarjan. Improved time bounds for the maximum flow problem. *SIAM Journal on Computing*, 18:939–954, 1989.

[4] R. J. Anderson and J. C. Setubal. *Network Flows and Matching: First DIMACS Implementation Challenge*, chapter Parallel and sequential implementations of maximum-flow algorithms. American Mathematical Society, 1993.

[5] K. M. Anstreicher and T. Terlaky. A monotonic build-up simplex algorithm for linear programming. *Operations Research*, 42:556–561, 1994.

[6] Dash Associates. Xpress-mosel language reference manual. `http://home.dei.polimi.it/malucell/didattica/appunti/mosel/mosel-language_reference.pdf`, 2003.

[7] M. Bacharach. Matrix rounding problems. *Management Science*, 12:732–742, 1966.

[8] Zs. Barta. Vasút optimalizálási problémák: matematikai módszerek és modellek. Master's thesis, Budapesti Műszaki és Gazdaságtudományi Egyetem, 2011.

[9] F. Bilen, Zs. Csizmadia, and T. Illés. Anstreicher-terlaky type monotonic simplex algorithms for linear feasibility problems. *Optimization Methods and Software*, 22:679–695, 2007.

[10] J. Cheriyan and S. N. Maheshwari. Analysis of preflow push algorithms for maximum network flow. *SIAM Journal on Computing*, 6:1057–1086, 1989.

[11] R. V. Cherkassky. Algorithm of construction of maximum flow in networks with complexity $\mathcal{O}(V^2E^{1/2})$ operations. *Mathematical Methods of Solution of Economical Problems*, 7:112–115, 1977.

[12] Zs. Csizmadia. *New pivot based methods in linear optimization, and an application in the petroleum industry.* PhD thesis, Eötvös Lóránd Tudományegyetem, 2007.

[13] Zs. Csizmadia and T. Illes. The s-monotone index selection rules for pivot algorithms of linear programming. *European Journal of Operational Research*, in press, 2012.

[14] G. B. Dantzig. *Activity Analysis of Production and Allocation*, chapter Maximization of a linear function of variables subject to linear inequalities, pages 339–347. Wiley, New York, 1951.

[15] G. B. Dantzig. *Linear Programming and Extensions.* University Press, Princeton, NJ, 1962.

[16] U. Derigs and W. Meier. Implementing goldberg's max-flow algorithm: A computational investigation. *Zeitschrift für Operations Research*, 33:383–403, 1989.

[17] E. A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Mathematics Doklady*, 11:1277–1280, 1970.

[18] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the Association of Computing Machinery*, 19:248–264, 1972.

[19] J. Egerváry. Mátrixok kombinatorikus tulajdonságairól. *Matematikai és Fizikai Lapok*, 38:16–28, 1931.

[20] ELTE. Library for efficient modeling and optimization in networks. `http://lemon.cs.elte.hu/trac/lemon`.

[21] L. R. Ford and D. R. Fulkerson. A simple algorithm for finding maximal network flows and an application to the hitchcock problem. *Research Memorandum RM-1604, The RAND Corporation, Santa Monica, California*, 1955.

[22] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[23] L. R. Ford and D. R. Fulkerson. A simple algorithm for finding maximal network flows and an application to the hitchcock problem. *Canadian Journal of Mathematics*, 9:210–218, 1957.

[24] A. Frank. *Connections in Combinatorial Optimization*. Oxford Lecture Series in Mathematics and its Applications 38. Oxford University Press, 2011.

[25] Z. Galil. An $\mathcal{O}(n^{5/3}m^{2/3})$ algorithm for the maximum flow problem. *Acta Informatica*, 14:221–242, 1980.

[26] Z. Galil and A. Namaad. An $\mathcal{O}(nm\log^2 n)$ algorithm for the maximum flow problem. *Journal of Computer and System Sciences*, 21:203–217, 1980.

[27] GCC. the GNU Compiler Collection, http://www.gnu.org/software/gcc/.

[28] A. V. Goldberg, M. D. Grigoriadis, and R. E. Tarjan. Efficiency of the network simplex algorithm for the maximum flow problem. *Mathematical Programming*, 50:277–290, 1991.

[29] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. *Journal of the ACM*, 35:921–940, 1988.

[30] D. Goldfarb and J. Hao. A primal simplex algorithm that solves the maximum flow problem in at most nm pivots and $o(n^2m)$ time. *Mathematical Programming*, 47:353–365, 1990.

[31] D. Goldfarb and J. Hao. On strongly polynomial variants of the network simplex algorithm for the maximum flow problem. *Operations Research Letters*, 10:383–387, 1991.

[32] D.M. Greig, B.T. Porteous, and A.H. Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society*, 51:271–279, 1989.

[33] T. E. Harris and F. S. Ross. Fundamentals of a method for evaluating rail net capacities. Technical report, Research Memorandum RM-1573, The RAND Corporation, Santa Monica, California, 1955.

[34] T. Illés, M. Makai, and Zs. Vaik. Railway engine assignment models based on combinatorial and integer programming. Technical report, Eötvös Loránd University of Sciences, Department of Operations Research, 2005.

[35] T. Illés, M. Makai, and Zs. Vaik. Combinatorial optimization model for railway engine assignment problem. In Leo G. Kroon and Rolf H. Möhring, editors, *5th Workshop on Algorithmic Methods and Models for Optimization of Railways*, 2006.

[36] A. Kamath and O. Palmon. Improved interior point algorithms for exact and approximate solution of multicommodity flow problems. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 502–511, 1995.

[37] S. Kapoor and P. M. Vaidya. Speeding up Karmarkar's algorithm for multicommodity flows. *Mathematical Programming*, 73:111–127, 1996.

[38] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[39] A. V. Karzanov. Determining the maximal flow in a network by the method of pre-flows. *Soviet Mathematics Doklady*, 15:434–437, 1974.

[40] Gy. Y. Katona, A. Recski, and Cs. Szabó. *A számítástudomány alapjai*. Typotex, 2006.

[41] L. G. Khachian. Polynomial algorithms in linear programming. *Zhurnal Vychislitelnoi Matematiki i Matematicheskoi Fiziki*, 20:53–72, 1980.

[42] E. Klafszky. *Hálózati folyamok*. Bólyai János Matematikai Társulat, 1969.

[43] D. König. Graphok és mátrixok. *Matematikai és Fizikai Lapok*, 38:116–119, 1931.

[44] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[45] E. Lawler. *Kombinatorikus Optimalizálás: hálózatok és matroidok*. Műszaki Kiadó, 1982.

[46] V. M. Malhotra, M. P. Kumar, and S. N. Maheshwari. An $\mathcal{O}(n^3)$ algorithm for finding maximum flows in networks. *Information Processing Letters*, 7:277–278, 1978.

[47] S. M. Murray. *An interior point approach to the generalized flow problem with costs and related problems*. PhD thesis, Stanford University, 1993.

[48] A. Schrijver. On the history of the transportation and maximum flow problems. *Mathematical Programming*, 91:437–445, 2002.

[49] A. Schrijver. Flows in railway optimization. *Nieuw Archief voor Wiskunde*, 5:126–131, 2008.

[50] Y. Shiloach. An $\mathcal{O}(nI \log^2 I)$ maximum flow algorithm. Technical report, Technical Report, STAN-CS-78-702, Computer Science Department, Stanford University, Stanford, CA, 1978.

[51] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26:362–391, 1983.

[52] T. Terlaky. A convergent criss-cross method. *Optimization: A Journal of Mathematical Programming and Operations Research*, 16:683–690, 1985.

[53] Z. M. Wang. A finite conformal-elimination free algorithm over oriented matroid programming. *Chinese Annals of Mathematics*, 8:120–125, 1987.