EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

Hálózati folyamok és alkalmazásaik

Hatékony algoritmusok a minimális költségű folyam feladatra

DIPLOMAMUNKA

Kovács Péter

programtervező matematikus hallgató

Témavezetők:

Király Zoltán
egyetemi docens
ELTE Természettudományi Kar
Számítógéptudományi Tanszék

Jüttner Alpár
tudományos segédmunkatárs
ELTE Természettudományi Kar
Operációkutatási Tanszék

Budapest
2007

Javított változat, 2012.02.17.

Tartalomjegyzék

Bevezetés

A hálózati folyamok az operációkutatás egyik legszélesebb körben alkalmazott eszköztára. Közvetlenül felhasználható telekommunikációs vagy egyéb kommunális hálózatok tervezési feladatainak megoldására, továbbá alapvető építőeleme különböző hálózati útvonalválasztó és forgalommenedzselő eljárásoknak, logisztikai problémák megoldására szolgáló algoritmusoknak is.

A különböző alkalmazások mindegyikében valamilyen entitást (terméket, adatot, villamos energiát stb.) szeretnénk továbbítani az adott hálózat egy vagy több pontjából más pontokba minél hatékonyabban. Ezzel kapcsolatban három alapvető problémakör fogalmazható meg: *legrövidebb út*, *maximális folyam* és *minimális költségű folyam*.

Jelen dolgozat célkitűzése a minimális költségű folyam feladat elméleti hátterének és különböző megoldási módszereinek áttekintése, valamint ezek közül néhány kiválasztott algoritmus hatékony megvalósításának bemutatása és összehasonlítása. Minden implementációt C++ nyelven, a LEMON hálózattervező és optimalizálási programkönyvtár keretein belül, a már meglévő eszközeinek felhasználásával végeztünk.

A tárgyalás során feltesszük, hogy az Olvasó rendelkezik a téma vizsgálatához szükséges matematikai ismeretekkel, tisztában van a gráfelmélet alapfogalmaival és szokásos jelöléseivel, továbbá ismeri az alapvető gráfelméleti algoritmusokat is.

1. Minimális költségű folyam

A minimális költségű folyam a hálózati folyamok legalapvetőbb modellje, amely – mint ahogy később látni fogjuk – speciális esetként magában foglal sok más folyamfeladatot is, többek között a két legismertebbet: a legrövidebb út és a maximális folyam problémát. A hálózati folyamok elméleti hátterét, a sokféle ismert algoritmust és ezek szerteágazó alkalmazási területeit nagyon jól bemutatja Ahuja, Magnanti és Orlin [2] műve.

1.1. A vizsgált feladat

A minimális költségű folyam feladat lényege egy olyan minimális költségű „szállítási terv", folyam meghatározása, amellyel az adott hálózat *termelő* csúcsaiból a *fogyasztó* csúcsokba eljuttathatjuk a megfelelő mennyiségű terméket oly módon, hogy az élekre vonatkozó kapacitáskorlátokat betartjuk.

Legyen $G = (V, E)$ egy irányított, gyengén összefüggő gráf $n = |V|$ csúccsal és $m = |E|$ éllel. Minden $(i, j) \in E$ élhez hozzárendelünk egy $l_{ij} \in \mathbb{R}$, $l_{ij} \geq 0$ *alsó* és egy $u_{ij} \in \mathbb{R} \cup \cup\{+\infty\}$, $u_{ij} \geq l_{ij}$ *felső* korlátot, valamint egy $c_{ij} \in \mathbb{R}$ *költséget*, amely az adott élen folyó egységnyi folyam költségét jelöli. Feltesszük tehát, hogy a hálózatban a folyam költsége lineárisan függ a mennyiségétől. Ezen kívül minden $i \in V$ csúcshoz hozzárendelünk egy $b(i) \in \mathbb{R}$ értéket, amely a termelését, illetve fogyasztását jelöli. Ha $b(i) > 0$, akkor i *termelő csúcs* $b(i)$ termeléssel, ha $b(i) < 0$, akkor i *fogyasztó csúcs* $-b(i)$ fogyasztással, ha pedig $b(i) = 0$, akkor i *közvetítő csúcs*. Ezen feltételek mellett szeretnénk meghatározni az egyes éleken az $x_{ij} \in \mathbb{R}$ folyamértékeket oly módon, hogy az összköltség minimális legyen. A legtöbb esetben azonban minden mennyiség- és költségérték egész, és a megoldást is egészértékű folyamként keressük.

Ezek alapján a vizsgált optimalizálási modell az alábbi módon formalizálható: határozzuk meg a

$$\min \sum_{(i,j)\in E} c_{ij} x_{ij} \tag{1}$$

értéket a

$$\sum_{j\,:\,(i,j)\in E} x_{ij} - \sum_{j\,:\,(j,i)\in E} x_{ji} = b(i) \qquad \forall i \in V, \tag{2}$$

$$l_{ij} \leq x_{ij} \leq u_{ij} \qquad \forall (i,j) \in E \tag{3}$$

feltételek mellett, ahol feltesszük, hogy $\sum_{i\in V} b(i) = 0$.

A probléma tehát egy lineáris programozási feladat. Legyen A a gráf $n \times m$-es ún. *csúcs-él szomszédsági mátrixa*, ahol minden oszlop egy $(i, j) \in E$ élhez tartozik: az i. sorában $+1$-et, a j. sorában -1-et tartalmaz, a többi érték pedig nulla. Ekkor a feladat mátrix alakban az alábbi módon írható fel: határozzuk meg a

$$\min cx \tag{4}$$

értéket az

$$Ax = b\,, \tag{5}$$

$$l \leq x \leq u \tag{6}$$

feltételek mellett.

A (2), illetve (5) feltételt *mennyiségkiegyenlítettségi feltételnek* (*mass balance constraint*) vagy *termelés/fogyasztás feltételnek* (*supply/demand constarint*) nevezzük, amely azt mondja ki, hogy minden $i \in V$ csúcshoz az összes kifolyó és az összes befolyó folyammennyiség különbsége egyezzen meg a $b(i)$ termeléssel. A (3), illetve (6) feltételt pedig *korlátfeltételnek* (*flow bound constraint*) nevezzük, amely tipikusan a hálózat fizikai tulajdonságait, illetve a folyammal kapcsolatos elvárásainkat jellemzi.

1.2. Speciális esetek

Ebben az alfejezetben bemutatjuk a minimális költségű folyam feladat néhány speciális esetét, amelyek többnyire önálló feladatként is ismertek, és gyakran előfordulnak különböző alkalmazásokban. Ezekkel szeretnénk szemléltetni, hogy a vizsgált probléma egy igen általános keretet határoz meg, amelyen belül sok speciálisabb feladat is megfogalmazható. Így a később bemutatásra kerülő megoldási módszerek széles körben, sok különböző területen alkalmazhatók.

1.2.1. Legrövidebb út

A *legrövidebb út* (*shortest path*) talán a legismertebb és legegyszerűbb folyamfeladat: keressünk egy legrövidebb (legkisebb költségű) utat a megadott hálózat egy s kezdőcsúcsából egy t célcsúcsába a c_{ij} élköltségek szerint. Az előző alfejezet jelöléseivel legyen $l_{ij} = 0$ és $u_{ij} = 1$ minden (i, j) élre, valamint $b(s) = 1$, $b(t) = -1$ és $b(i) = 0$ minden más i csúcsra. Ekkor a kapott optimális megoldás egy egységnyi folyamot visz át s-ből t-be egy legrövidebb út mentén. Amennyiben pedig egy s csúcsból kiindulva az összes többi csúcsba keressük a legrövidebb utat, akkor legyen $b(s) = n - 1$ és $b(i) = -1$ minden más i csúcsra, továbbá minden u_{ij} kapacitás legyen legalább $n-1$. Ekkor a megoldás az s csúcsból egy egységnyi folyamot visz át minden más csúcsba egy-egy legrövidebb út mentén.

1.2.2. Maximális folyam

A *maximális folyam* (*maximum flow*) feladat bizonyos értelemben a legrövidebb út komplementer modelljének is tekinthető. Az utóbbinál ugyanis csak az élek költsége játszik szerepet, ezzel szemben a maximális folyam feladatban nincsenek költségek, csak kapacitáskorlátokkal foglalkozunk. A feladat az, hogy egy adott s forrás csúcsból egy t nyelő

csúcsba találjunk egy maximális értékű megengedett folyamot (vagyis olyan folyamot, amely nem sérti meg az u_{ij} kapacitáskorlátokat). Ezt formalizálhatjuk minimális költségű folyamként oly módon, hogy $b(i) = 0$ minden i csúcsra, és $c_{ij} = 0$ minden $(i, j) \in E$ élre, továbbá kiegészítjük a gráfot egy (t, s) éllel, amelyre $c_{ts} = -1$, $l_{ts} = 0$, $u_{ts} = \infty$. Ekkor egy optimális megoldás nyilván maximalizálja a folyamot a (t, s) élen, ehhez viszont ennek a folyammennyiségnek az eredeti éleken el kell jutnia s-ből t-be (hiszen minden $b(i) = 0$). Így a megoldás az eredeti gráfon egy maximális s–t folyamot ad.

1.2.3. Szállítási feladat

A *szállítási feladat* (*transportation problem*) a minimális költségű folyam olyan speciális esete, ahol a csúcsok V halmaza felbontható két (nem feltétlenül azonos méretű) V_1, V_2 halmazra oly módon, hogy minden $i \in V_1$ termelő csúcs, minden $j \in V_2$ fogyasztó csúcs, továbbá minden $(i, j) \in E$ élre $i \in V_1$ és $j \in V_2$ (vagyis G páros gráf). Egyes alkalmazásokban kapacitáskorlátokat sem feltételeznek, ilyenkor $l_{ij} = 0$, $u_{ij} = \infty$ minden élre.

1.2.4. Hozzárendelési feladat

A *hozzárendelési feladatban* (*assignment problem*) adott két azonos méretű halmazunk: V_1 és V_2, valamint a lehetséges hozzárendelések egy $E \subseteq V_1 \times V_2$ halmaza és mindegyikhez egy c_{ij} költség. A feladat az, hogy minden V_1-beli elemhez rendeljünk pontosan egy V_2-belit egy $(i, j) \in E$ él mentén oly módon, hogy az összköltség minimális legyen. Ezt a feladatot is formalizálhatjuk minimális költségű folyamként: legyen $G = (V_1 \cup V_2, E)$, ahol $b(i) = 1$ minden $i \in V_1$ csúcsúra, $b(j) = -1$ minden $j \in V_2$ csúcsra, és minden élre $l_{ij} = 0$, $u_{ij} = 1$.

1.2.5. Áramfeladat

Az *áramfeladat* (*circulation problem*) a minimális költségű folyam olyan speciális esete, amelyben csak közvetítő csúcsok vannak, vagyis minden $b(i) = 0$, a teljes folyammennyiség körbeáramlik a hálózatban.

Ehhez példaként tekinthetünk egy olyan alkalmazást, amelyben egy optimális menetrendet szeretnénk megadni egy repülőgép társaság gépei számára. Ebben a modellben minden gép a különböző városok repülőterei között jár körbe. Minden l_{ij} alsó korlátot egyre választunk azokban az esetekben, ha biztosítanunk kell egy járatot az i csomópontból a j csomópontba. Itt a gráf csúcsai a térbeli elhelyezkedés és az indulási/érkezési időpont kombinációját reprezentálják, például Budapest 10:00, London 11:00. Az egyes repülőtereken való várakozásnak pedig további élek feleltethetők meg, amelyeknek szintén lehetnek kapacitáskorlátai (a repülőtér befogadó képessége) és nullától különböző költségei is.

1.3. Általánosítások

Az alábbiakban áttekintjük a minimális költségű folyam feladat szokásos általánosításait és azok néhány alkalmazási lehetőségét. Ezekkel jelen dolgozat keretében részletesebben nem foglalkozunk, azonban az itt felsorolt példák szintén jól mutatják a vizsgált modell jelentőségét, továbbá az általánosabb folyamproblémák részfeladataiként gyakran kapunk minimális költségű folyam feladatot.

1.3.1. Konvex költségű folyam

Az eddig tárgyalt modellben feltettük, hogy a folyam költsége a mennyiség lineáris függvénye, ezzel szemben a *konvex költségű folyam* (*convex cost flow*) feladatban csak annyit kötünk meg, hogy ez valamilyen konvex függvény legyen. Számos olyan probléma van, amelynél szükséges lehet ez az általánosítás, például: elektromos hálózatokban az ellenállások hatására bekövetkező villamosenergia-veszteségek leírása; az egyes vonalak terheltsége, illetve az esetleges torlódások miatt fellépő késleltetés, várakozási idő kezelése különböző közlekedési és kommunikációs hálózatokban.

1.3.2. Általánosított folyam

Az *általánosított folyam* (*generalized flow*) modellben az egyes élek „fogyaszthatják" és „növelhetik" is a rajtuk átfolyó folyamot. Ha az i csúcsból az (i, j) élen x_{ij} egységnyi folyam lép ki, akkor a j csúcsba $\mu_{ij}x_{ij}$ egységnyi érkezik meg. Ha $\mu_{ij} < 1$, akkor az él *veszteséges*, ha pedig $\mu_{ij} > 1$, akkor *nyereséges*. Ez a megközelítés természetes módon felmerül sok alkalmazásban, például: villamos energia továbbítása elektromos hálózatokban, ahol a vezetékeken veszteség léphet fel; vízvezeték-hálózatokban, csatornákban a szivárgás, illetve párolgás jelenségének kezelése; valamilyen romlandó termék szállítása; valamint pénzügyi, gazdasági rendszerek modellezése, ahol az egyes élek reprezentálják a különböző befektetési lehetőségeket, eseményeket, a hozzájuk rendelt szorzó tényezők pedig a várható hozamot, illetve veszteséget.

1.3.3. Többtermékes folyam

A *többtermékes folyam* (*multicommodity flow*) feladatban egyszerre több árucikk van jelen ugyanabban a hálózatban. Mindegyiknek külön vannak termelő és fogyasztó csúcsai, külön kiegyenlítettségi feltételek vonatkoznak rájuk, viszont az élek kapacitásain osztoznak. Az alapvető probléma tehát nyilván az, hogyan osszuk fel a rendelkezésre álló kapacitásokat az egyes termékek között. Ez a feladat is számos valós probléma modellezésére alkalmas, például: utasok közlekedése egy város bizonyos pontjai között; különböző termékek, mezőgazdasági termények szállítása egy-egy országon belül, illetve

országok között; valamint üzenetek küldése egy kommunikációs hálózatban különböző küldő-címzett párok között.

1.4. Történeti előzmények

Elsőként L. V. Kantorovich orosz matematikus vizsgált olyan szállítási problémákat, amelyek a minimális költségű folyam modell kialakulásához vezettek. 1939-ben írt értekezése [43] csak jóval később, az 1960-as években vált ismertté, miután nyugaton is elkezdődtek hasonló kutatások. 1975-ben T. C. Koopmans holland matematikussal megosztva Közgazdasági Alfred Nobel-emlékdíjjal jutalmazták.

Kantorovich kilenc különböző problémaosztályt határozott meg, amelyek között szerepel például: munka elosztása az egyes munkagépek között; rendelések elosztása gyárak, termelőüzemek között; különböző nyersanyagok, illetve üzemanyag elosztása; valamint optimális teherszállítási tervek készítése. Később, az 1940-es években megjelent munkáiban már foglalkozott kapacitáskorláttal rendelkező hálózatokkal is, valamint felismerte a primál és duál feladat kapcsolatát.

Nyugaton F. L. Hitchcock foglalkozott először hasonló problémákkal, 1941-ben megfogalmazta az 1.2.3. szakaszban bemutatott klasszikus szállítási feladatot [40]. Egy $m \times n$ dimenziós geometriai reprezentációt adott m gyár és n város közti áruszállítás problémájára, bevezette a megengedett tartomány és az extremális pontok fogalmát. 1947-ben Koopmans pedig megfogalmazott egy optimalitási kritériumot, és megmutatta, hogy az extremális pontoknak a gráf terminológiában feszítőfák felelnek meg.

H. W. Kuhn 1955-ben [51] a hozzárendelési feladat megoldására kidolgozott egy algoritmust, amely két magyar matematikus, Egerváry Jenő és Kőnig Dénes eredményeire épült, ezért az ő tiszteletükre *magyar módszernek* nevezte el. Később ez az operációkutatás egyik legjelentősebb módszere lett, amely hálózati folyamokkal, párosításokkal és matroidokkal kapcsolatos számos algoritmus kidolgozását alapozta meg.

A hálózati folyamok – és ezen belül a minimális költségű folyam modell – ma használt fogalomrendszerének kialakulása, az alapvető elméleti eredmények jelentős része L. R. Ford és D. R. Fulkerson munkásságának köszönhető. 1962-ben megjelent értekezésükben [22] számos eredményüket összefoglalták, megfogalmazták és részletesen elemezték többek között a minimális költségű folyam problémát, és bemutattak több különböző megoldási módszert is.

Az 1960-as évektől előtérbe került ez a problémaosztály, és az 1990-es évekig folyamatosan születtek újabb és újabb eredmények. Ezek részletes időrendi áttekintésétől azonban eltekintünk, helyette a 2. fejezetben az egyes megoldási módszerek tárgyalásakor utalunk azok történeti előzményeire. Ahuja, Magnanti és Orlin [2] könyvében megtalálható minden ismertebb algoritmus történeti hátterének összefoglalása számos hivatkozással, Schrijver [59] cikke pedig a hálózati folyamok elméletének fejlődését mutatja be.

1.5. Feltételezések és átalakítások

A dolgozat további részeiben az 1.1. alfejezetben leírt minimális költségű folyam feladatot vizsgáljuk, és arra keresünk megoldó algoritmusokat. Ehhez azonban először megfogalmazunk néhány további feltételezést, és megmutatjuk, hogy megfelelő átalakításokkal ezek biztosíthatók.

1.1. Feltételezés. *Minden adat (kapacitás, élköltség és termelés/fogyasztás érték) egész.*

Ez a feltételezés az alkalmazások többségénél nem jelent lényegi megszorítást, ugyanis a racionális számokat egy kellően nagy számmal megszorozva egésszé alakíthatjuk, az irracionális számokat pedig eleve valamilyen racionális közelítéssel helyettesítjük a számítógépes ábrázolás során. A 3.2. alfejezetben megmutatjuk, hogy ha ezen feltétel mellett a feladatnak van optimális megoldása, akkor létezik olyan optimális megoldás is, amelyben minden x_{ij} érték egész, ezért a továbbiakban minden esetben ilyen folyamot keresünk.

1.2. Feltételezés. *Minden kapacitásérték véges.*

Általában ez a feltétel sem jelent megszorítást, ugyanis a végtelen kapacitásokat a többi adat függvényében lecserélhetjük kellően nagy véges értékekre. Olyankor viszont ez nem alkalmazható, amikor az eredeti feladatban a végtelen kapacitások miatt a megengedett megoldásokhoz tartozó célfüggvényértékek halmaza alulról nem korlátos, ugyanis a transzformált feladatnak nyilván ilyenkor is létezne optimális megoldása. Az ilyen eseteket azonban könnyen kizárhatjuk az alábbi kritérium alapján [35].

1.3. Tétel. *A minimális költségű folyam feladatnak akkor és csak akkor létezik optimális megoldása, ha létezik megengedett megoldása, és a G gráf nem tartalmaz olyan negatív összköltségű irányított kört, amelyben minden él kapacitása végtelen.*

Tehát ha az eredeti feladatban szerepelnek végtelen kapacitások, akkor megvizsgáljuk, hogy létezik-e végtelen kapacitású negatív kör. Ha igen, akkor a feladatnak nincs optimális megoldása, egyébként pedig a végtelen kapacitásokat lecserélhetjük kellően nagy véges kapacitásokra.

Vegyük észre, hogy az 1.1. és 1.2. feltételezések mellett a feladatnak csak véges sok különböző (egészértékű) megengedett megoldása lehet, ezért ha létezik megengedett megoldás, akkor létezik optimális megoldás is.

1.4. Feltételezés. *A $b(i)$ termelés/fogyasztás értékek összege nulla.*

Ez a kikötés mindenképpen szükséges ahhoz, hogy a feladatnak legyen megengedett megoldása, ugyanis:

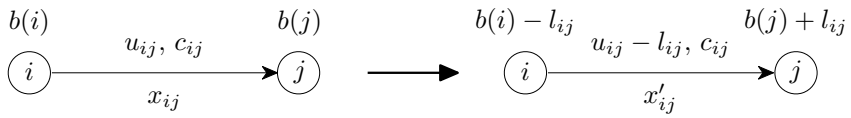
$$\sum_{i\in V} b(i) = \sum_{i\in V} \Big(\sum_{j\,:\,(i,j)\in E} x_{ij} \;-\; \sum_{j\,:\,(j,i)\in E} x_{ji} \Big) =$$
$$= \sum_{i\in V} \sum_{j\,:\,(i,j)\in E} x_{ij} \;- \sum_{i\in V} \sum_{j\,:\,(j,i)\in E} x_{ji} \;=\; 0.$$

1.5. Feltételezés. *Minden l_{ij} alsó korlát nulla.*

Ez a kikötés az általánosság megszorítása nélkül feltehető, ugyanis minden minimális költségű folyam feladat ekvivalens módon átalakítható úgy, hogy ez teljesüljön. Legyen (i, j) egy tetszőlegesen rögzített él $l_{ij} \neq 0$ alsó korláttal. Ekkor az 1. ábrán látható módon ezen az élen az x_{ij} folyamérték helyett tekintsük az $x'_{ij} = x_{ij} - l_{ij}$ értéket, és legyen $l'_{ij} = 0$, $u'_{ij} = u_{ij} - l_{ij}$. Ezáltal a mennyiségkiegyenlítettségi feltételben $b(i)$ l_{ij}-vel csökken, $b(j)$ pedig l_{ij}-vel növekszik, így ezt az átalakítást minden élre elvégezve adódik, hogy:

$$b'(i) = b(i) - \sum_{j\,:\,(i,j)\in E} l_{ij} + \sum_{j\,:\,(j,i)\in E} l_{ji}.$$

A fenti transzformációt egy két fázisból álló megoldási módszerként is értelmezhetjük. Először minden (i, j) élen l_{ij} folyamot folyatunk, aztán a második lépésben az ehhez hozzáadandó további folyammennyiségeket (az x'_{ij} értékeket) keressük oly módon, hogy azok összköltsége minimális legyen.



1. ábra. Nem nulla alsó korlátok eltávolítása

Könnyen látható, hogy az így definiált l'_{ij}, u'_{ij}, x'_{ij} és $b'(i)$ értékekre a szükséges feltételek teljesülnek, továbbá az eredeti és a transzformált feladat megoldásai kölcsönösen egyértelműen megfeleltethetők egymásnak oly módon, hogy a két célfüggvényérték közti különbség minden esetben megegyezik a $\sum_{(i,j)\in E} c_{ij} l_{ij}$ konstanssal. Tehát a két feladat ekvivalens.

Ezek alapján a továbbiakban eltekintünk az l_{ij} alsó korlátoktól, csak az u_{ij} kapacitásokkal foglalkozunk.

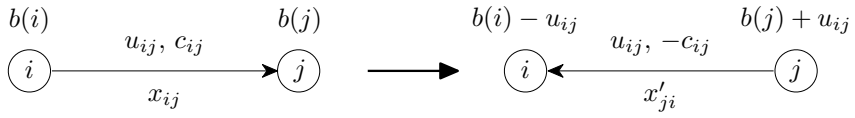
1.6. Feltételezés. *Minden c_{ij} élköltség nemnegatív.*

Az előzőhöz hasonlóan ez a feltétel is biztosítható a feladat ekvivalens átalakításával. A szükséges transzformáció lényege, hogy minden negatív költségű (i, j) él esetén az x_{ij} változót $u_{ij} - x'_{ji}$-re cseréljük a feladatban. Ennek megfelelően a 2. ábrán látható módon az élt megfordítjuk, lecseréljük egy (j, i) élre $c'_{ji} = -c_{ij}$ költséggel és $u'_{ji} = u_{ij}$ kapacitással.

Ezt az átalakítást is egy két lépcsős megoldási módszerként értelmezhetjük. Először minden negatív költségű (i, j) élen küldjük át a lehető legtöbb, vagyis u_{ij} egységnyi folyamot (ezáltal $b(i)$ u_{ij}-vel csökken, $b(j)$ pedig u_{ij}-vel növekszik), a második lépésben pedig a megfordított éleken visszafolyó mennyiségek (az x'_{ji} értékek) azt adják meg, hogy mennyit kell ebből levonni, mekkora az a folyammennyiség, amelyet mégsem folyatunk át ezeken az éleken. Ezt a transzformációt minden élre elvégezve adódik, hogy:

$$b'(i) = b(i) - \sum_{j\,:\,(i,j)\in E'} u_{ij} + \sum_{j\,:\,(j,i)\in E'} u_{ji},$$

ahol E' jelöli az eredeti gráf negatív költségű éleinek halmazát.



2. ábra. Negatív élköltségek eltávolítása

Megjegyezzük, hogy a fenti átalakítás következtében a gráfban megjelenhetnek párhuzamos élek, akkor is, ha eredetileg nem voltak. Az ilyen feladatok esetén azonban a gyakorlatban többnyire olyan gráfimplementációt használnak, amely megfelelően kezeli a párhuzamos éleket. A továbbiakban mindig ilyen megvalósítást feltételezünk, viszont az elméleti tárgyalás során az egyszerűség kedvéért továbbra is megtartjuk az élek (i, j) jelölését.

1.7. Feltételezés. *G irányított gráf.*

Az eddigiekben is irányított gráfra fogalmaztuk meg a feladatot, viszont bizonyos alkalmazásokban irányítatlan hálózatok szerepelnek. Az alábbiakban megmutatjuk, hogy az 1.5. és 1.6. feltételezések mellett minden irányítatlan gráfra megfogalmazott minimális költségű folyam feladatot természetes módon átírhatunk irányított gráfra. Fontos megjegyeznünk azonban, hogy irányítatlan hálózatok esetén ez a két feltétel lényegi megszorítást jelent. A fent bemutatott transzformációk ilyenkor nem alkalmazhatók, ugyanis nem tudjuk meghatározni, hogy a kérdéses éleken kezdetben átküldendő l_{ij}, illetve u_{ij} folyammennyiséget hogyan osszuk el a két irány között. Ráadásul nem nulla alsó korlátok, illetve negatív élköltségek esetén az irányítatlan feladatnak általában úgy van értelme, ha kikötjük, hogy minden élen csak egy irányba folyhat folyam, ez viszont NP-nehéz problémákat eredményez [27].

Feltesszük tehát, hogy nincsenek alsó korlátok, és minden élköltség nemnegatív. Jelölje $\{i, j\}$ az i és j csúcsok közötti irányítatlan élt (vagyis $\{i, j\} = \{j, i\}$), amelynek kapacitása $u_{\{i,j\}}$, költsége pedig $c_{\{i,j\}} \geq 0$. Ekkor az $\{i, j\}$ élen a két irányba összesen $u_{\{i,j\}}$ folyam folyhat, az egységnyi költség pedig mindkét esetben $c_{\{i,j\}}$. A folyam fogalma tehát

szükségszerűen irányított marad, ezért a megoldást x_{ij}, x_{ji} ($\{i,j\} \in E$) alakban keressük. Ezek alapján irányítatlan gráfra az alábbi módon írható fel a feladat: határozzuk meg a

$$\min \sum_{\{i,j\}\in E} \left(c_{\{i,j\}} x_{ij} + c_{\{i,j\}} x_{ji}\right) \tag{7}$$

értéket a

$$\sum_{j\,:\,\{i,j\}\in E} x_{ij} - \sum_{j\,:\,\{i,j\}\in E} x_{ji} = b(i) \qquad \forall i \in V, \tag{8}$$

$$x_{ij} + x_{ji} \le u_{\{i,j\}} \qquad \forall \{i,j\} \in E, \tag{9}$$

$$0 \le x_{ij}, x_{ji} \qquad \forall \{i,j\} \in E \tag{10}$$

feltételek mellett.

Mivel azonban feltettük, hogy a költségek nemnegatívak, ha létezik optimális megoldás, akkor nyilván létezik olyan optimális megoldás is, ahol minden $\{i,j\}$ élre x_{ij} és x_{ji} közül legalább az egyik nulla. A (8) feltétel ugyanis átírható a következő módon:

$$\sum_{j\,:\,\{i,j\}\in E} x_{ij} - \sum_{j\,:\,\{i,j\}\in E} x_{ji} = \sum_{j\,:\,\{i,j\}\in E} (x_{ij} - x_{ji}) = b(i) \qquad \forall i \in V.$$

Tehát x_{ij} és x_{ji} közül a kisebb helyett nullát, a nagyobb helyett a kettő különbségét írva a (8) – (10) feltételek továbbra is teljesülnek, a (7) célfüggvény értéke pedig nem növekszik. Ezzel azt értük el, hogy a megoldás *átfedésmentes* lesz, vagyis bármely élen legfeljebb az egyik irányba folyik folyam. Vegyük észre, hogy pozitív alsó korlátok esetén ez az átalakítás nem lenne lehetséges.

Ezek után írjuk át a (7) – (10) feladatot irányított gráfra oly módon, hogy minden $\{i,j\}$ irányítatlan élnek egy (i,j) és egy (j,i) élt feleltetünk meg $u_{\{i,j\}}$ kapacitással és $c_{\{i,j\}}$ költséggel. Ekkor a két modell nem átfedő folyamai nyilván kölcsönösen egyértelműen megfeleltethetők egymásnak (azonos célfüggvényértékkel), az átfedő folyamok pedig mindkét esetben olyan nem átfedő folyamokká alakíthatók, amelyek költsége nem nagyobb. Tehát az egyik feladat bármely optimális megoldásából könnyen kaphatunk optimális megoldást a másikra, és viszont, vagyis a két probléma ekvivalens.

Az eddigiekben adott feltételezéseket összegezve fogalmazzuk meg újra a vizsgált feladatot. Határozzuk meg a

$$\min \sum_{(i,j)\in E} c_{ij} x_{ij} \tag{11}$$

értéket a

$$\sum_{j\,:\,(i,j)\in E} x_{ij} - \sum_{j\,:\,(j,i)\in E} x_{ji} = b(i) \qquad \forall i \in V, \tag{12}$$

$$0 \le x_{ij} \le u_{ij} \qquad \forall (i,j) \in E \tag{13}$$

feltételek mellett, ahol feltesszük, hogy minden $b(i)$, c_{ij}, u_{ij} és x_{ij} érték egész, minden c_{ij} költség nemnegatív, valamint $\sum_{i\in V} b(i) = 0$. A továbbiakban minimális költségű folyam feladat alatt minden esetben ezt értjük.

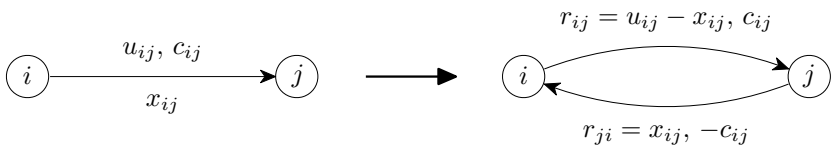
1.6. Élköltségek kezelése

Ebben az alfejezetben még néhány szükséges fogalmat tárgyalunk, amelyek az élköltségekkel rendelkező folyammodellhez kapcsolódnak.

1.6.1. Maradék hálózat

A hálózati folyamokkal kapcsolatban az egyik alapvető fogalom a *maradék* (*reziduális*) kapacitás, illetve hálózat, amelyeket ismertnek feltételezünk. Ebben a modellben azonban – a maximális folyam feladattal ellentétben – költségeket is értelmezünk a maradék hálózat élein.

A maradék hálózat fogalmának lényege az, hogy a folyamokat nem abszolút értelemben vizsgáljuk, hanem egy rögzített folyamhoz viszonyítva, annak módosításaként, ami az algoritmusokban tipikusan egy köztes megoldáson való javításnak felel meg. Legyen tehát x egy rögzített folyam a G hálózatban. Ekkor egy (i, j) élen i-ből j-be még $u_{ij} - x_{ij}$ folyamot küldhetünk át c_{ij} költséggel, j-ből i-be pedig a már meglévő folyam csökkentésével átvihetünk x_{ij} mennyiséget. Az utóbbi esetben nyilván csökkentjük a célfüggvény értékét, így ebben az irányban $-c_{ij}$-re választjuk az élköltséget.



3. ábra. Maradék hálózat

Ezek alapján az eredeti gráf minden (i, j) élét a 3. ábrán látható módon egy (i, j) és egy (j, i) élre cseréljük. Az (i, j) él maradék kapacitása $r_{ij} = u_{ij} - x_{ij}$, költsége c_{ij}, a (j, i) él maradék kapacitása pedig $r_{ji} = x_{ij}$, költsége $-c_{ij}$. Az ilyen módon definiált élek közül a pozitív maradék kapacitással rendelkezők alkotják az x folyamhoz tartozó G_x maradék hálózatot.

Megjegyezzük, hogy ha a G gráfban két csúcs között mindkét irányba vezet él, akkor a G_x maradék hálózatban megjelenhetnek párhuzamos élek, amelyeknek maradék kapacitása, illetve költsége különböző lehet. Ezért – ahogy korábban is említettük – mindig olyan megvalósítást feltételezünk, amelyben ez nem okoz gondot, viszont többnyire megtartjuk az élek kissé pontatlan (i, j) jelölését.

1.6.2. Redukált élköltségek

A minimális költségű folyam feladatra adott algoritmusokban gyakran van szükség arra, hogy a gráf éleinek költségét a végpontjaikhoz hozzárendelt ún. *potenciálértékek* alapján

módosítsuk. Ezek az egyes algoritmusokban tipikusan valamilyen köztes számítási eredményeket jelenítenek meg, amelyeket a különböző tudományterületeken definiált potenciálokhoz hasonló módon értelmezhetünk.

A G gráf minden i csúcsához hozzárendelve egy tetszőleges előjelű $\pi(i)$ potenciálértéket, a π-hez tartozó *redukált élköltségeket* az eredeti költség és az él végpontjai között lévő potenciálkülönbség összegeként definiáljuk:

$$c_{ij}^{\pi} = c_{ij} - \pi(i) + \pi(j) \qquad \forall (i,j) \in E.$$

Azonosan nulla potenciálfüggvény esetén nyilván az eredeti költségeket kapjuk vissza.

Mivel sok algoritmusban alkalmazzuk ezeket, fontos megvizsgálni az eredeti és a redukált élköltségek kapcsolatát.

1.8. Állítás. *Tetszőleges π potenciálfüggvény esetén teljesülnek az alábbiak.*

a) Az eredeti és a redukált élköltségek szerint vett minimális költségű folyam feladat optimális megoldásai megegyeznek, és a két célfüggvényérték közti különbség minden esetben a $\sum_{i \in V} \pi(i)b(i) = \pi b$ konstans.

b) Minden W irányított körre $\sum_{(i,j) \in W} c_{ij}^{\pi} = \sum_{(i,j) \in W} c_{ij}$.

c) Minden P irányított útra, amely a k csúcsból az l csúcsba visz $\sum_{(i,j) \in P} c_{ij}^{\pi} = \\ = \sum_{(i,j) \in P} c_{ij} - \pi(k) + \pi(l)$.

Bizonyítás. Az első összefüggés belátásához induljunk ki az azonosan nulla potenciálfüggvényből. Ha egy i csúcs potenciálját $\pi(i)$-re változtatjuk, akkor definíció szerint minden i-ből induló élen $\pi(i)$-vel csökken, és minden i-be mutató élen $\pi(i)$-vel növekszik a redukált költség. Így tetszőleges megengedett folyam esetén a célfüggvényértékben történt változás a $\pi(i)$ érték, valamint az i csúcs összes kifolyó és befolyó folyama különbségének szorzata. Ez az utóbbi különbség viszont éppen a $b(i)$ termelés/fogyasztás érték, tehát a fenti lépést minden csúcsra elvégezve adódik az állítás *a)* pontja.

A *b)* és *c)* összefüggések pedig egyszerűen következnek abból, hogy az egymáshoz csatlakozó irányított élek redukált költségeinek összegében az ellentétes előjellel szereplő csúcspotenciálok kiejtik egymást. ■

1.7. Optimalitási feltételek

Az alábbiakban bemutatunk három különböző, de ekvivalens optimalitási feltételt a minimális költségű folyam feladatra. Ezek több szempontból is nagyon fontos szerepet töltenek be a megoldó algoritmusokban: egyrészt egy konkrét megoldás esetén könnyen és hatékonyan ellenőrizhetők, másrészt közvetlenül is ötletet adnak egy-egy módszer kidolgozásához. Először azonban kimondunk egy fontos tételt, amelyből ezek a feltételek következnek.

1.9. Tétel. *A minimális költségű folyam feladat egy tetszőleges x megengedett megoldásából egy tetszőleges y megengedett megoldás előállítható a G_x maradék hálózat legfeljebb m irányított köre mentén történő változtatással, és az y folyam költsége megegyezik az x folyam költségének és a körök mentén küldött folyamok költségének összegével.*

A tétel bizonyítása azon a tulajdonságon alapul, hogy minden megengedett folyam szétbontható egyszerű utakra és körökre [2].

1.10. Tétel (Negatív kör optimalitási feltétel). *A minimális költségű folyam feladat egy x^* megengedett megoldása akkor és csak akkor optimális, ha a G_{x^*} maradék hálózat nem tartalmaz negatív összköltségű irányított kört.*

Bizonyítás. Tegyük fel, hogy egy x megengedett megoldáshoz tartozó G_x maradék hálózat tartalmaz negatív kört. Ekkor x nyilván nem lehet optimális, ugyanis a célfüggvényérték csökkenthető azáltal, hogy pozitív folyammennyiséget küldünk egy ilyen kör mentén. Vagyis ha x^* optimális, akkor G_{x^*} nem tartalmaz negatív kört. Másrészt tegyük fel, hogy egy x^* megengedett megoldás esetén G_{x^*} nem tartalmaz negatív kört, és legyen x' egy optimális megoldás. Az 1.9. tétel szerint ekkor az $x' - x^*$ folyam előáll G_{x^*}-beli irányított körök összegeként. Mivel azonban G_{x^*}-ban nincs negatív kör, ez a módosítás biztosan nem csökkenti a célfüggvény értékét, így $cx^* \leq cx'$, tehát x^* is optimális megoldás. ■

Az egyes algoritmusokban gyakran használunk csúcspotenciálokat és redukált költségeket, ezért sokszor hasznos a fenti tétel alábbi átfogalmazása.

1.11. Tétel (Redukált költség optimalitási feltétel). *A minimális költségű folyam feladat egy x^* megengedett megoldása akkor és csak akkor optimális, ha létezik olyan π potenciálfüggvény, amelyre a G_{x^*} maradék hálózat minden (i, j) élének redukált költsége nemnegatív.*

A feltétel szemléletes jelentése érthetőbbé válik, ha összevetjük a legrövidebb út probléma egyszerűbb optimalitási kritériumával. A szokásos jelöléssel legyen $d(i)$ az i csúcs távolságcímkéje. Egy megoldás csak akkor lehet optimális, ha minden (i, j) élre

$$d(j) \leq d(i) + c_{ij}.$$

Vegyük észre, hogy ezt átírhatjuk az alábbi ekvivalens formába: minden (i, j) élre

$$c_{ij}^d = c_{ij} + d(i) - d(j) \geq 0.$$

Ez az alak pedig $\pi = -d$ választással a fenti optimalitási feltételnek felel meg: a G_{x^*} maradék hálózat minden (i, j) élére

$$c_{ij}^\pi = c_{ij} - \pi(i) + \pi(j) \geq 0.$$

Általában is igaz, hogy szoros kapcsolat áll fenn a minimális költségű folyam feladatban szereplő csúcspotenciálok és megfelelő távolságcímkék között. A $\mu(i) = -\pi(i)$ mennyiséget értelmezhetjük úgy, mint egy egységnyi terméknek az i csúcsban való beszerzési költségét, és így c_{ij}^{π} a teljes költsége annak, hogy egy egységnyi terméket az i csúcsban megvásárolunk, majd a j csúcsba elszállítjuk és ott eladjuk. Ezek alapján egy szállítási terv, egy folyam nyilván akkor és csak akkor optimális, ha azon (i, j) élek mentén, amelyek maradék kapacitása pozitív (még szállíthatunk), már nem tudunk javítani, vagyis ha $c_{ij}^{\pi} \geq 0$, azaz $\mu(j) \leq \mu(i) + c_{ij}$. Ez pedig éppen a fenti optimalitási kritériumot adja.

Bizonyítás. A redukált költség optimalitási feltétel ekvivalens a negatív kör feltétellel. Tegyük fel, hogy egy x^* megengedett megoldásra és π potenciálra teljesül az előbbi. Ekkor nyilván minden G_{x^*}-beli W irányított körre $\sum_{(i,j)\in W} c_{ij}^{\pi} \geq 0$, így az 1.8. állításban szereplő *b)* összefüggést felhasználva $\sum_{(i,j)\in W} c_{ij} \geq 0$, tehát G_{x^*}-ban nincs negatív kör. Másrészt tegyük fel, hogy G_{x^*} nem tartalmaz negatív kört, és egészítsük ki ezt a gráfot egy olyan virtuális s csúccsal, amelyből minden valós i csúcsba egy nulla költségű él vezet. Az így kapott gráf szintén nem tartalmaz negatív kört, és az s csúcsból minden csúcs elérhető. Ismert, hogy ilyenkor az egyes csúcsoknak az s csúcstól mért $d(\cdot)$ távolsága jóldefiniált ($d(i) \leq 0$ minden i csúcsra), és minden (i, j) élre $d(j) \leq d(i) + c_{ij}$, vagyis $c_{ij} + d(i) - d(j) \geq 0$. Tehát $\pi = -d$ választással éppen a kívánt feltételt kapjuk. ■

Az előző két tételben a maradék hálózat segítségével adtuk meg az optimalitás feltételét, az alábbiakban pedig kimondjuk ezek ekvivalens átfogalmazását az eredeti hálózatra nézve.

1.12. Tétel (Complementary Slackness optimalitási feltétel). *A minimális költségű folyam feladat egy x^* megengedett megoldása akkor és csak akkor optimális, ha létezik olyan π potenciálfüggvény, amellyel a G gráf minden (i, j) élére teljesülnek az alábbiak.*

a) Ha $c_{ij}^{\pi} > 0$, akkor $x_{ij}^* = 0$.

b) Ha $0 < x_{ij}^* < u_{ij}$, akkor $c_{ij}^{\pi} = 0$.

c) Ha $c_{ij}^{\pi} < 0$, akkor $x_{ij}^* = u_{ij}$.

Bizonyítás. Felhasználva, hogy a maradék hálózat definíciója szerint egy $(i, j) \in E$ él esetén $c_{ji}^{\pi} = -c_{ij}^{\pi}$, továbbá hogy az (i, j) él pontosan akkor szerepel a G_{x^*} hálózatban, ha $x_{ij}^* < u_{ij}$, a (j, i) él pedig pontosan akkor, ha $x_{ij}^* > 0$, egyszerű meggondolással adódik, hogy a tétel ekvivalens a redukált költség feltétellel. ■

1.8. A feladat dualitása

A lineáris programozásban különösen is fontos fogalom a *dualitás*, amelyet alapvetően ismertnek feltételezünk. Minden lineáris programozási feladathoz megadható egy ún. duál feladat, amelynek optimális célfüggvényértéke megegyezik az eredeti (primál) probléma optimális megoldásának célfüggvényértékével (ha ez létezik). Maximalizálási feladat duálisa minimalizálási feladat, és fordítva.

Sok olyan probléma van, amelyet megfogalmazhatunk lineáris programozási feladatként, de a specialitásait kihasználva hatékonyabb megoldási módszereket adhatunk rá, mint az általános esetre. Ilyen a minimális költségű folyam feladat is. A dualitás viszont ezekben az esetekben is fontos, ugyanis a duál probléma megtalálása szinte mindig együtt jár egy polinomiális megoldási módszer megtalálásával, továbbá többnyire egyszerű és hatékonyan ellenőrizhető optimalitási feltételeket eredményez. Az alábbiakban az 1.5. alfejezetben megfogalmazott (11) – (13) feladat dualitását vizsgáljuk.

Egy lineáris programozási feladat duálisának meghatározásakor egy duál változót rendelünk a primál feladat minden korlátozó feltételéhez (kivéve a nemnegativitási feltételeket). Az i csúcsra vonatkozó termelés/fogyasztás feltételhez rendeljük hozzá a $\pi(i)$ változót, az (i, j) élre vonatkozó kapacitásfeltételhez pedig az α_{ij} változót. Ezekkel a jelölésekkel a duális minimális költségű folyam feladat a következőképpen írható fel: határozzuk meg a

$$\max \sum_{i \in V} b(i)\pi(i) - \sum_{(i,j) \in E} u_{ij}\alpha_{ij} \tag{14}$$

értéket a

$$\pi(i) - \pi(j) - \alpha_{ij} \leq c_{ij} \qquad \forall (i, j) \in E, \tag{15}$$

$$\alpha_{ij} \geq 0 \qquad \forall (i, j) \in E \tag{16}$$

feltételek mellett.

A további vizsgálatok egyszerűbbé tehetők azáltal, ha ebből az alakból az α_{ij} változókat kitranszformáljuk. Vegyük észre, hogy a $\pi(i)$ értékeket csúcspotenciáloknak tekintve, és az 1.6.2. szakaszban bevezetett redukált költségek $c^{\pi}_{ij} = c_{ij} - \pi(i) + \pi(j)$ jelölését felhasználva, a (15) feltételt átírhatjuk az alábbi módon:

$$\alpha_{ij} \geq -c^{\pi}_{ij} \qquad \forall (i, j) \in E. \tag{17}$$

Mivel a célfüggvény értékét maximalizálni szeretnénk, és minden $\alpha_{ij} \geq 0$ változó együtthatója negatív vagy nulla, mindegyikhez a legkisebb lehetséges értéket szeretnénk hozzárendelni. Így a (16) és (17) feltételek alapján

$$\alpha_{ij} = \max\{0, -c^{\pi}_{ij}\} \qquad \forall (i, j) \in E. \tag{18}$$

Tehát ha ismerjük a $\pi(i)$ változók optimális értékét, akkor az α_{ij} változók optimális értéke meghatározható ilyen módon. Ezek alapján a (14) – (16) duál feladatból kitranszformálva

az α_{ij} változókat az alábbi ekvivalens feladatot kapjuk:

$$\max \sum_{i \in V} b(i)\pi(i) - \sum_{(i,j) \in E} u_{ij} \max\{0, -c_{ij}^{\pi}\}. \tag{19}$$

Ezzel a duál probléma optimális $\pi(i)$ értékek megtalálására egyszerűsödött.

Ezek után kimondunk két fontos tételt, amelyek a lineáris programozásban ismert általános dualitás tételek speciális esetei [2], a bizonyításukat viszont jelen dolgozat keretében nem tárgyaljuk.

1.13. Tétel (Gyenge dualitás tétel). *Legyen $z(x)$ a primál feladat egy x megengedett megoldásához tartozó célfüggvényérték, $w(\pi)$ pedig a duál feladat egy π megengedett megoldásához tartozó célfüggvényérték. Ekkor $w(\pi) \leq z(x)$.*

A tétel közvetlen következménye, hogy ha egy x primál megengedett megoldás és egy π duál megengedett megoldás célfüggvényértéke megegyezik, akkor mindkettő optimális. A következő tétel pedig ilyen primál–duál megoldáspárok létezését mondja ki.

1.14. Tétel (Erős dualitás tétel). *Ha a primál feladatnak létezik optimális megoldása, akkor a duál feladatnak is, és a két optimális célfüggvényérték megegyezik.*

Megjegyezzük, hogy a korábban adott 1.1. és 1.2. feltételezések mellett a tételben elegendő lenne megengedett megoldás létezését feltenni, ugyanis abból következik az optimális megoldás létezése.

Fontos kapcsolat van a minimális költségű folyam feladat dualitása és a korábban bemutatott optimalitási feltételek között, amelyet az alábbi tételben fogalmazunk meg.

1.15. Tétel. *Egy x megengedett folyam és π potenciálfüggvény esetén $z(x) = w(\pi)$ akkor és csak akkor teljesül, ha az (x, π) pár kielégíti a redukált költség optimalitási feltételt.*

Tehát az 1.11. és 1.12. tétel értelmében optimális potenciálfüggvények megegyeznek az optimális duál megoldásokkal. Így a későbbiekben tárgyalt minden olyan algoritmus, amely redukált élköltségekkel dolgozik, és a folyamértékek mellett megfelelő csúcspotenciálokat is meghatároz, egyaránt szolgáltat optimális primál és duál megoldást is.

1.9. Optimális folyamok és potenciálok

Az előző két alfejezetben bemutattuk az optimális folyamok és optimális csúcspotenciálok kapcsolatát. Az alábbiakban pedig azt a kérdést vizsgáljuk, hogyan lehet egy optimális folyamból kiindulva optimális potenciálfüggvényt, illetve optimális potenciálfüggvényből kiindulva optimális folyamot meghatározni.

Elsőként tegyük fel, hogy adott egy x^* optimális folyam. Ekkor az 1.11. tétel bizonyításában leírt módon egy legrövidebb út-kereséssel meghatározhatunk optimális csúcspotenciálokat.

Ezek után vizsgáljuk meg a fordított esetet, amikor optimális potenciálértékek ismeretében szeretnénk egy optimális folyamot meghatározni. Ehhez a G gráf éleit soroljuk be három csoportba a c_{ij}^{π} redukált költségek szerint.

- Ha $c_{ij}^{\pi} > 0$, akkor az 1.12. optimalitási feltétel *a)* pontja szerint x_{ij}^*-nek nullának kell lennie. Legyen tehát $x_{ij}^* = 0$, és töröljük az (i, j) élt a gráfból.
- Ha $c_{ij}^{\pi} < 0$, akkor az 1.12. optimalitási feltétel *c)* pontja szerint legyen $x_{ij}^* = u_{ij}$, és töröljük az (i, j) élt a gráfból. Mivel azonban u_{ij} egységnyi folyamot átküldtünk ezen az élen, $b(i)$-t csökkentsük, $b(j)$-t pedig növeljük u_{ij}-vel.
- Ha $c_{ij}^{\pi} = 0$, akkor az 1.12. optimalitási feltétel alapján x_{ij}^* tetszőleges értéket felvehet 0 és u_{ij} között.

Az ilyen módon kapott hálózatban egy megengedett folyamot keresve az eredeti feladat egy optimális megoldását kapjuk. A 3.2.1. szakaszban pedig megmutatjuk, hogy a megengedett folyam-keresés visszavezethető maximális folyam feladatra.

2. Megoldási módszerek

Az elmúlt néhány évtized során számos algoritmust dolgoztak ki a minimális költségű folyam feladat megoldására. Ezeket többféleképpen is csoportosíthatjuk: egyrészt a megközelítés módja szerint megkülönböztetünk *primál*, *duál*, *primál–duál* és *relaxációs* módszereket, másrészt a műveletigényük alapján beszélhetünk *kvázipolinomiális*, *polinomiális* és *erősen polinomiális* algoritmusokról. Ebben a fejezetben az utóbbi, elméleti és történeti szempontból is meghatározóbb csoportosítás szerint áttekintjük az ismert módszerek széles skáláját.

2.1. Műveletigény-elemzés

Az egyes megoldási módszerek tárgyalása során központi szerepet kap a műveletigényük elemzése és aszimptotikus összehasonlítása, amelyben mindig a maximális lépésszám (legrosszabb eset) vizsgálatára szorítkozunk.

A minimális költségű folyam feladatot megoldó algoritmusok műveletigényét a következő paraméterek függvényében adjuk meg: n a gráf csúcsainak, m az éleinek száma, C az élköltségek, U pedig a kapacitások és termelés/fogyasztás értékek felső korlátja. Azt mondjuk, hogy egy algoritmus *kvázipolinomiális*, ha a műveletigénye n, m, C, U egy polinomiális függvényével felülről becsülhető; *polinomiális*, ha a futási ideje az élköltségek és kapacitások nagyságának csak logaritmusától függ, vagyis n, m, $\log C$, $\log U$ függvényében polinomiális; illetve *erősen polinomiális*, amennyiben egy kizárólag n-től és m-től függő polinomiális függvénnyel felülről becsülhető.

A vizsgált módszerekben gyakran merülnek fel részfeladatként legrövidebb út, illetve maximális folyam problémák, így a futási idejük erősen függ attól, hogy ezek megoldására milyen algoritmust használunk. Ezért ilyenkor a műveletigényt ennek függvényében adjuk meg, az alábbiakban pedig áttekintjük az említett két feladatra jelenleg ismert legjobb lépésszámkorlátokat. Jelölje $SP(n, m, C)$ a legrövidebb út-keresés műveletigényét egy C-nél nem nagyobb nemnegatív egész élköltségeket tartalmazó gráfban, $MF(n, m, U)$ pedig a maximális folyam-keresés műveletigényét egy U-nál nem nagyobb egész kapacitásokkal rendelkező hálózatban.

Nemnegatív élköltségek esetén a legrövidebb út probléma a Dijkstra-algoritmus Fibonacci-kupacokat használó változatával [23] megoldható

$$SP(n, m) = O(m + n \log n) \tag{20}$$

erősen polinomiális időben. Viszonylag kicsi, egész élköltségek esetén azonban ennél jobb korlátok is adhatók. Ahuja, Mehlhorn, Orlin és Tarjan 1990-ben [3] adott algoritmusa $O(m + n\sqrt{\log C})$ időben fut, a legjobb ismert polinomiális korlátok pedig a Thorup által 2003-ban [66] bemutatott adatszerkezetekkel érhetők el:

$$SP(n, m, C) = O(m + n \min\{\log\log n, \log\log C\}). \tag{21}$$

A maximális folyam feladatra Dinic 1970-ben [20] egy $O(n^2m)$ idejű algoritmust, tőle függetlenül Edmonds és Karp pedig 1972-ben [21] egy $O(nm^2)$ idejű algoritmust adott, amelyek Ford és Fulkerson módszerén [22] alapultak. Dinic algoritmusa a Sleator és Tarjan által 1983-ban [61] erre a célra kifejlesztett dinamikus fákkal $O(nm \log n)$ időben megvalósítható.

Goldberg és Tarjan 1986-ban [31] kidolgozott egy másik nagyon hatékony megoldási módszert, az *előfolyam* (*preflow*) algoritmust, és dinamikus fák felhasználásával egy $O(nm \log(n^2/m))$ idejű implementációt adtak rá. Később Ahuja, Orlin és Tarjan [4] hasonló hatékonyságú skálázó algoritmusokat dolgozott ki.

A legjobb erősen polinomiális korlát King, Rao és Tarjan [47] eredménye:

$$MF(n, m) = O(nm \log_{m/(n \log n)} n). \tag{22}$$

A legjobb polinomiális korlátot pedig 1998-ban Goldberg és Rao [30] adta:

$$MF(n, m, U) = O(m \min\{n^{2/3}, \sqrt{m}\} \log(n^2/m) \log U). \tag{23}$$

2.2. Kvázipolinomiális algoritmusok

Ebben az alfejezetben a minimális költségű folyam problémára adott kvázipolinomiális algoritmusokat mutatjuk be. Ezek nem csupán történeti jelentőséggel bírnak, ugyanis egyrészt a 2.2.1. és 2.2.2. szakaszban bemutatott *negatív kör* és *ismételt legrövidebb út* módszerek adják a két alapvető megközelítési módot (primál és duál), így alapját képezik több bonyolultabb algoritmusnak is; másrészt a 2.2.5. és 2.2.6. szakaszban tárgyalt *relaxációs* és *szimplex* alapú algoritmusok a gyakorlatban a leghatékonyabb módszerek közé tartoznak.

2.2.1. Negatív kör módszer

A *negatív kör* (*negative cycle* vagy *cycle-canceling*) módszert Klein [48] dolgozta ki 1967-ben. Az algoritmus az 1.10. negatív kör optimalitási feltételen alapul: először meghatározunk egy megengedett megoldást, majd minden iterációban negatív köröket keresünk az aktuális folyamhoz tartozó maradék hálózatban. Ha találunk ilyen kört, akkor azon a legkisebb maradék kapacitású élének megfelelő mennyiségű folyamot folyatunk, és ezzel az élt kiiktatjuk a maradék hálózatból, ha pedig már nem találunk, akkor az említett feltétel értelmében a folyam optimális.

Klein egy $O(nm)$ idejű távolságcímkéző legrövidebb út-algoritmust használt a negatív körök keresésére. Mivel a kezdeti folyam összköltségének mCU nyilván felső korlátja, és az algoritmus ezt minden lépésben legalább eggyel javítja (ugyanis minden kapacitás és folyamérték egész), az iterációk száma is legfeljebb mCU, így a teljes műveletigény $O(nm^2CU)$.

Az általános módszer nem határozza meg a negatív körök kiválasztásának sorrendjét, speciális választási szabályok alkalmazásával később különböző polinomiális változatokat dolgoztak ki. Weintraub [68] azt javasolta, hogy minden iterációban olyan kör mentén történjen javítás, amellyel a legnagyobb változás érhető el, később Barahona és Tardos Éva [6] ennek egy módosított változatáról megmutatta, hogy $O(m^2 \log(mCU)\, SP(n, m, C))$ időben fut. Goldberg és Tarjan [33] pedig egy erősen polinomiális negatív kör algoritmust adott, amelyben mindig minimális átlagköltségű körök mentén javítanak. Megjegyezzük továbbá, hogy a 2.2.6. szakaszban bemutatott primál hálózati szimplex algoritmus is a negatív kör módszer egy sajátos változata.

A negatív kör algoritmusok primál megközelítési módot valósítanak meg, aminek fontos előnye, hogy bármely megengedett folyamból kiindulva alkalmazhatók. Ez különösen akkor lehet hasznos, ha adott egy közel optimális megoldás, és azt akarjuk javítani, amire tipikus példa az, amikor egy hálózatban valamilyen módszerrel már meghatároztunk egy minimális költségű folyamot, viszont utána néhány él költsége megváltozott. Ilyenkor a meglévő folyamot ezzel a módszerrel többnyire kevés iteráció árán optimális folyammá alakíthatjuk, feltéve persze, hogy a hálózatban történt változás kevés élt érint és nem túl jelentős.

A negatív kör módszer egy egyszerűbb, valamint a fent említett erősen polinomiális változatát is megvalósítottuk, elsősorban elméleti jelentőségük miatt. A 3.2. alfejezetben ezt a két algoritmust és implementációjuk módját mutatjuk be.

2.2.2. Ismételt legrövidebb út módszer

Jewell 1958-ban [42], Iri 1960-ban [41], valamint Busacker és Gowen 1960-ban [13] egymástól függetlenül dolgozta ki az *ismételt legrövidebb út* (*successive shortest path*) duál módszert. Megmutatták, hogy a minimális költségű folyam feladat megoldható legrövidebb út-keresések sorozatával, és ezzel egy $O(n^2 mU)$ idejű algoritmust adtak. Később Edmonds és Karp [21], valamint Tomizawa felismerte, hogy csúcspotenciálok alkalmazásával biztosítani lehet a nemnegatív élköltségeket a legrövidebb út-keresésekhez, ezt felhasználva pedig egy nagyságrendileg gyorsabb, $O(nU\, SP(n, m, nC))$ idejű algoritmust dolgoztak ki (a továbbiakban minden esetben ezt vizsgáljuk). Megjegyezzük, hogy a futási időben azért $SP(n, m, nC)$ szerepel, mert az algoritmusban használt redukált élköltségeknek nem C, hanem nC a felső korlátja.

Az előző szakaszban bemutatott negatív kör módszerben egy primál megengedett megoldásból indulunk ki, és a célfüggvény lépésenkénti javításával érjük el, hogy teljesüljön az optimalitás feltétele. Ezzel szemben az ismételt legrövidebb út módszer duál megközelítési módot alkalmaz: az azonosan nulla folyamból kiindulva végig egy olyan folyamot tartunk nyilván, amelyre teljesülnek a nemnegativitási és kapacitásfeltételek, viszont megsértheti a termelés/fogyasztás feltételeket. Minden iterációban kiválasztunk egy s csúcsot, ahol még többlet van (vagyis a termelése nagyobb, mint a kifolyó és befolyó folyam

különbsége), és egy t csúcsot, ahol még hiány van, majd a maradék hálózat egy legrövidebb útja mentén átküldjük a lehető legnagyobb folyammennyiséget s-ből t-be. A fent említett csúcspotenciálok alkalmazásával biztosítani lehet, hogy az algoritmus minden lépésében teljesüljön az 1.11. feltétel, tehát az eredeti élköltségek helyett használhatjuk a nemnegatív redukált költségeket, és így Dijkstra algoritmusával sokkal hatékonyabban kereshetünk legrövidebb utakat. Mivel minden lépésben csökkentjük valamely csúcsban a többletet, egy másik csúcsban pedig a hiányt, az iterációk száma legfeljebb $nU/2$ lehet, tehát az algoritmus valóban $O(nU\, SP(n, m, nC))$ időben fut.

A későbbiekben bemutatunk olyan algoritmusokat, amelyek általában jóval hatékonyabbak, azonban viszonylag kis termelés/fogyasztás értékek esetén ez a módszer az egyik leggyorsabb. A gyakorlatban pedig sokszor felmerül például az a feladat, hogy egy hálózat két csúcsa között k értékű minimális költségű folyamot keresünk valamely kis k egészre, továbbá az 1.2.4. szakaszban tárgyalt hozzárendelési feladat is ilyen hálózatot eredményez ($U = 1$). Ezen kívül – ahogy korábban is említettük – ez a módszer alapját képezi több bonyolultabb, skálázási technikát alkalmazó algoritmusnak is.

Egyszerűsége és hatékonysága miatt megvalósítottuk az ismételt legrövidebb út algoritmust, valamint annak egy kapacitásskálázó változatát is, amelyeket a 3.3. alfejezetben részletesen tárgyalunk.

2.2.3. Primál–duál módszer

Ford és Fulkerson [22] kidolgozott egy *primál–duál* megoldási módszert a minimális költségű folyam feladatra. Ez az ismételt legrövidebb út algoritmushoz hasonlóan egy olyan folyamot tart nyilván, amely megsértheti a termelés/fogyasztás feltételeket, és ezen minden lépésben legrövidebb utak mentén küldött folyamokkal javít. A különbség az, hogy a primál–duál módszer minden iterációban egy maximális folyam feladatot megoldva egyszerre több legrövidebb út mentén javít.

Az algoritmus azzal indul, hogy kiegészítjük a G gráfot egy s forrás csúccsal és egy t nyelő csúccsal oly módon, hogy minden i termelő csúcshoz felveszünk egy (s, i) élt $b(i)$ kapacitással, és minden i fogyasztó csúcshoz egy (i, t) élt $-b(i)$ kapacitással. Az egyes iterációkban pedig először egy s-ből indított legrövidebb út-kereséssel meghatározzuk a csúcspotenciálokat, hogy teljesüljön a redukált költség feltétel, majd a maradék hálózatban egy maximális folyamot keresünk s-ből t-be kizárólag nulla redukált költségű éleket felhasználva. Mivel minden redukált költség nemnegatív, a talált folyamban megjelenő $s \rightsquigarrow t$ utak biztosan legrövidebb utak. Ezen folyam mentén javítva legalább eggyel csökken az s csúcsban a többlet, továbbá legalább eggyel csökken a t csúcs potenciálja is (hiszen a következő legrövidebb út-keresés során már legalább egy távolságra lesz s-től). Mivel kezdetben az összes többlet legfeljebb $nU/2$ lehet, és egyik csúcs potenciálja sem csökkenhet $-nC$ alá, legfeljebb $\min\{nU/2, nC\}$ iteráció lehet, így az algoritmus műveletigénye $O\big(\min\{nU, nC\}(SP(n, m, nC) + MF(n, m, U))\big)$.

A módszer onnan kapta a nevét, hogy a lineáris programozásban ismert általános primál–duál megközelítési módot alkalmazza. Egy megengedett duál megoldást (csúcspotenciálok) és egy olyan primál megoldást tartunk nyilván, amely megsértheti a mennyiségkiegyenlítettségi feltételt, viszont teljesül rájuk a complementary slackness kritérium. Ezután minden lépésben először a lehető legtöbbet javítunk a primál megoldáson, majd módosítjuk a duál megoldást. Ez a primál–duál elv sok kombinatorikus optimalizálási problémára, sőt az általános lineáris programozási feladatra is alkalmazható, és többnyire igen hatékony algoritmusokat eredményez.

2.2.4. Out-of-Kilter módszer

Yakovleva 1959-ben, Minty 1960-ban és Fulkerson 1961-ben [25] egymástól függetlenül dolgozta ki az ún. *out-of-kilter* algoritmust, később Aashtiani és Magnanti pedig kifejlesztette ennek egy $O(mU\,SP(n, m, nC))$ idejű változatát. A módszer az előzőekhez hasonlóan bizonyos feltételeket folyamatosan fenntart, a többit pedig az egyes iterációk során legrövidebb út-keresésekkel igyekszik elérni. Itt azonban egy megengedett folyamból kiindulva éppen a mennyiségkiegyenlítettségi feltételt biztosítjuk mindvégig, a kapacitáskorlátokat és az optimalitási kritériumot viszont nem.

Az algoritmus minden élhez egy *kilter* számot rendel, amely azt a minimális értéket jelöli, amellyel növelve vagy csökkentve az élen folyó folyam nagyságát teljesül a kapacitásfeltétel és a complementary slackness optimalitási feltétel. Ezután minden iterációban egy legrövidebb út-kereséssel meghatározott kör mentén javítunk, és ezáltal csökkentjük valamely él kilter számát, egészen addig, amíg már nem létezik pozitív kilter számmal rendelkező (out-of-kilter) él.

2.2.5. Relaxációs módszer

1985-ben Bertsekas [9], majd 1988-ban Bertsekas és Tseng [10] kifejlesztett *relaxációs* algoritmusokat, amelyek az előzőekben bemutatott primál–duál módszer speciális változatai voltak. Mélyreható számítási elemzéseket végeztek, és közzétettek hatékony FORTRAN implementációkat RELAX, illetve RELAXT néven. Az ő eredményeik, továbbá Grigoriadis [38], valamint Kennington és Wang vizsgálatai szerint a gyakorlatban ezek az algoritmusok és a következő szakaszban tárgyalt hálózati szimplex módszerek általában jóval hatékonyabbnak bizonyultak a meglévő többi implementációnál, annak ellenére, hogy a legrosszabb esetben csak nagyságrendileg gyengébb lépésszámkorlátok adhatók rájuk. Azokban az esetekben viszont, amikor a hálózatban kis termelés/fogyasztás értékek szerepeltek, az ismételt legrövidebb út módszer volt a leggyorsabb. Ezen számítási tesztek döntő többségét Klingman, Napier és Stutz NETGEN [49] programjával generált véletlen hálózatokon végezték.

A relaxációs módszer az egészértékű programozási feladatok megoldására használt Lagrange relaxáció elvét alkalmazza. Ennek értelmében az i csúcsra vonatkozó ter-

melés/fogyasztás feltételt beszorozzuk egy $\pi(i)$ értékkel, és az eredményt kivonjuk a célfüggvényből. Az így kapott relaxált feladat a következő: rögzített π potenciálfüggvény esetén határozzuk meg a

$$w(\pi) \;=\; \min_{x} \sum_{(i,j)\in E} c_{ij}x_{ij} + \sum_{i\in V} \pi(i)\left(b(i) - \sum_{j\,:\,(i,j)\in E} x_{ij} + \sum_{j\,:\,(j,i)\in E} x_{ji}\right) \tag{24}$$

értéket a

$$0 \le x_{ij} \le u_{ij} \qquad \forall (i,j) \in E \tag{25}$$

feltételek mellett. A redukált élköltségekre bevezetett jelöléssel a fenti célfüggvény egyszerűbb formában is felírható:

$$w(\pi) \;=\; \min_{x} \sum_{(i,j)\in E} c^{\pi}_{ij}x_{ij} + \sum_{i\in V} \pi(i)b(i) \tag{26}$$

Mindkét alakból közvetlenül adódik, hogy azonosan nulla π függvény esetén az eredeti célfüggvényt kapjuk vissza.

Az algoritmusban egy π potenciálfüggvényt és ehhez egy, a fenti értelemben optimális x folyamot tartunk nyilván, amely nem feltétlenül megengedett. Belátható, hogy ilyenkor teljesül a redukált költség optimalitási feltétel. Ezután minden lépésben a π potenciált megtartva az x folyamot változtatjuk oly módon, hogy az optimalitási feltétel továbbra is teljesüljön, és legalább egy csúcsban csökkenjen a többlet, vagy módosítjuk a π függvényt, és ehhez keresünk optimális x folyamot, amelyekre $w(\pi)$ nagyobb, mint az előző iterációban volt. Ha már egyik csúcsban sincs többlet, akkor x egy optimális megoldása az eredeti feladatnak.

2.2.6. Hálózati szimplex módszer

G. B. Dantzig, aki az általános lineáris programozási feladatra a szimplex módszert kidolgozta, 1951-ben ennek speciális változataként bemutatta az első *hálózati szimplex* (*network simplex*) algoritmust a kapacitáskorlátok nélküli szállítási feladatra, később pedig ezt általánosította a minimális költségű folyam feladatra.

Ez az algoritmus tehát az eddigiektől merőben eltérő megközelítési módot alkalmaz: a szimplex módszer elvét valósítja meg a gráf terminológiában, közvetlenül a hálózaton. Ennek alapját az a felismerés adja, miszerint a minimális költségű folyam probléma – mint lineáris programozási feladat – bázis megoldásainak feszítőfák feleltethetők meg. A (primál) hálózati szimplex algoritmus működése során mindvégig egy alkalmas feszítőfaszerkezetet tartunk nyilván, amely meghatároz egy megengedett folyamot, és az egyes lépésekben ennek célfüggvényértékén próbálunk javítani. Ehhez kiválasztunk egy olyan élt, amelyet a fához hozzávéve kialakul egy negatív költségű kör, majd ezen kör mentén javítva a fa egyik élét kicseréljük a belépő élre. Belátható, hogy megfelelő feltételek biztosítása mellett ez az algoritmus véges sok lépésben optimális megoldást eredményez.

A hálózati szimplex módszer mai népszerűségét az 1970-es években vívta ki, amikor feszítőfák kezeléséhez hatékony indexelési módszereket dolgoztak ki. Az első ilyen faindexeket Johnson javasolta 1966-ban, majd Srinivasan és Thompson 1973-ban [62], valamint Glover, Karney, Klingman és Napier 1974-ben [28] megvalósította ezek különböző változatait. Az általuk kidolgozott implementációk, továbbá Bradley, Brown és Graves [12] hálózati szimplex algoritmusa a gyakorlatban hatékonyabbnak bizonyult a meglévő többi algoritmusnál. 1977-ben Helgason és Kennington [39], majd 1980-ban Armstrong, Klingman és Whitman [5] kidolgozott duál hálózati szimplex módszereket is, de ezekkel nem tudtak olyan jó eredményeket elérni, mint a primál változatokkal.

Zadeh 1973-ban [69] olyan szállítási feladatokat mutatott be, melyekre az akkor ismert negatív kör, ismételt legrövidebb út, primál–duál, out-of-kilter és hálózati szimplex implementációk mindegyike exponenciálisan sok iterációt végez (kellően nagy kapacitások és költségek esetén). Zadeh eredményeinek jelentősége elsősorban az volt, hogy bizonyos tesztadatok ennyire különböző algoritmusok számára egyaránt legrosszabb esetet jelenthetnek.

Több nyilvános hálózati szimplex implementáció is született FORTRAN nyelven. Ezek közül a legismertebbek Kennington és Helgason NETFLOW kódja [46], valamint Grigoriadis RNET kódja [38], amelyek később számos különböző algoritmus számára viszonyítási alapot képeztek.

Sok kutatás irányult polinomiális hálózati szimplex algoritmusok kidolgozására. Goldfarb és Hao [36], valamint Tarjan [64] olyan polinomiális primál változatokat adtak, amelyekben az egyes iterációk során megengedett a célfüggvény értékének növekedése is, Orlin, Plotkin és Tardos pedig kidolgoztak különböző polinomiális duál szimplex algoritmusokat [53, 57, 56]. Végül Orlin [55] mutatta be az első olyan polinomiális primál változatot, amelyben az iterációk során a célfüggvény értéke végig monoton csökkenően változik.

Ahogy korábban is említettük, a hálózati szimplex a gyakorlatban leggyorsabb módszerek közé tartozik, ezért az egyik legfontosabb célkitűzésünk volt ennek hatékony megvalósítása, amelyet a 3.4. alfejezetben tárgyalunk.

2.3. Polinomiális algoritmusok

A minimális költségű folyam feladatra adott legtöbb polinomiális algoritmus valamilyen skálázáson alapul. Ez a technika számos kombinatorikus optimalizálási problémára alkalmazható, és gyakran javíthatunk vele a megoldási módszerek hatékonyságán. Erre a feladatra J. Edmonds és R. M. Karp [21] vezette be a skálázás elvét 1972-ben, és ezzel az első polinomiális algoritmust adták. A következő években kevés kutatás folyt ezen a területen, viszont az 1980-as évektől egyre inkább felismerték a skálázás elméleti és gyakorlati jelentőségét, a ma ismert legjobb aszimptotikus korlátokat adó algoritmusok pedig szinte kivétel nélkül alkalmazzák.

A skálázó algoritmusok a feladatban adott feltételrendszer helyett annak egy módosított, valamilyen Δ paraméterrel jellemezhető mértékben gyengített változatából indulnak ki, és ennek megfelelő közelítő megoldások sorozatán keresztül haladnak egy optimális megoldás felé. Az első lépésben Δ értékét kellően nagyra választva – például $\Delta = U$ vagy $\Delta = C$ esetén – könnyen adhatunk olyan kezdeti megoldást, amelyre teljesülnek a gyengített optimalitási feltételek. Az egyes iterációk során pedig Δ értékét fokozatosan csökkentjük, és az aktuális megoldást egy jobb közelítő megoldássá alakítjuk. Ez az elv merőben különböző algoritmusokat eredményez attól függően, hogy mely feltételeket gyengítjük, illetve hogy az egyes lépésekben milyen módon javítjuk a közelítő megoldásokat. A minimális költségű folyam feladatra két alapvető skálázási módot alkalmaznak: a *kapacitásskálázást* (*capacity scaling* vagy *right-hand scaling*) és a *költségskálázást* (*cost scaling*).

2.3.1. Kapacitásskálázó algoritmusok

Edmonds és Karp fent említett algoritmusa az ismételt legrövidebb út módszer kapacitásskálázó változata volt. Ebben az algoritmusban a feladat optimalitási feltételeit kétféleképpen is gyengítjük: egyrészt az aktuális folyam megsértheti az egyes csúcsokra vonatkozó termelés/fogyasztás feltételeket, másrészt a maradék hálózat tartalmazhat negatív kört. Az egyes iterációkban pedig a Δ paraméter aktuális értéke mellett olyan legrövidebb utak mentén javítunk, amelyeken legalább Δ folyammennyiséget küldhetünk át. Ha már nem létezik ilyen út, akkor Δ értékét felezve a következő iterációba lépünk, illetve $\Delta < 1$ esetén az algoritmus befejeződik. Ezzel a skálázással a legrövidebb út-keresések száma $O(nU)$-ról $O(m \log U)$-ra csökken, így az algoritmus $O(m \log U \, SP(n, m, nC))$ időben fut.

Ez a módszer egyszerűsége ellenére elméleti és gyakorlati szempontból is a leghatékonyabb algoritmusok közé tartozik, ezért fontosnak tartottuk, hogy implementáljuk. A 3.3.2. szakaszban részletesen ismertetjük az általunk megvalósított változatát.

2.3.2. Költségskálázó algoritmusok

A költségskálázás módszerét Röck 1980-ban [58], majd ettől függetlenül Bland és Jensen 1985-ben [11] javasolta. Mindkét algoritmus maximális folyam-keresések sorozatával $O(n \log C \, MF(n, m, U))$ időben oldotta meg a feladatot. Goldberg és Tarjan 1987-ben [32] az ϵ-optimalitás fogalmát felhasználva – amelyet Bertsekas [8] és Tardos Éva [67] egymástól függetlenül fogalmazott meg – nagyságrendileg tudta javítani Röck algoritmusát, és ezzel $O(\min\{n^3, n^{5/3}m^{2/3}, nm \log n\} \log(nC))$ lépésszámkorlátot adtak. Később kidolgoztak még jó néhány hatékony implementációt és heurisztikát [29], és egy $O(nm \log(n^2/m) \log(nC))$ idejű változatot is bemutattak [34].

Goldberg és Tarjan költségskálázó algoritmusa is a feladat feltételrendszerének gyengítésén alapul. Az egyes iterációkban a Δ szerepét betöltő $\epsilon > 0$ paraméter aktuális értéke

mellett olyan folyamot állítunk elő, amelyre a maradék hálózat minden élének redukált költsége nagyobb vagy egyenlő, mint $-\epsilon$. Ehhez minden lépésben hasonló pumpáló és átcímkéző (push/relabel) műveleteket hajtunk végre, mint amilyeneket a maximális folyam feladatra adott előfolyam algoritmus végez. Ezután ϵ-t felezve a következő iterációba lépünk, illetve $\epsilon < \frac{1}{n}$ esetén optimális megoldást kapunk, ugyanis ekkor az 1.8. állítás *b)* pontja alapján a maradék hálózat minden W irányított körére $\sum_{(i,j)\in W} c_{ij} = \sum_{(i,j)\in W} c^{\pi}_{ij} \geq -\epsilon n > -1$, és mivel minden élköltség egész, a kör összköltsége nemnegatív, tehát teljesül az 1.10. optimalitási feltétel.

2.3.3. Egyéb polinomiális algoritmusok

Az előző két szakaszban ismertetett algoritmusokon kívül léteznek egyéb polinomiális módszerek is. Ahuja, Goldberg, Orlin és Tarjan 1988-ban [1] egy olyan algoritmust dolgozott ki, amely ötvözte a kapacitás- és költségskálázást, és dinamikus fák felhasználásával egy $O(nm \log\log U \log(nC))$ idejű hatékony implementációt adtak rá. Ezen kívül a negatív kör és a hálózati szimplex módszernek is vannak polinomiális változatai, amelyeket a 2.2.1. és 2.2.6. szakaszban tárgyaltunk.

2.4. Erősen polinomiális algoritmusok

Miután 1972-ben Edmonds és Karp bemutatta az első polinomiális algoritmust a minimális költségű folyam feladatra, még sokáig nyitott kérdés maradt, hogy létezik-e erősen polinomiális megoldás. Ennek elméleti jelentőségét többek között az adja, hogy irracionális költségek és kapacitások esetén is alkalmazható. Elsőként Tardos Éva [67] adott ilyen algoritmust, később viszont ezt sok más erősen polinomiális módszer követte. Orlin 1984-ben [53], valamint Fujishige 1986-ban [24] $O(m^2 \log n\, SP(n,m))$ idejű, Galil és Tardos pedig 1986-ban [26] $O(n^2 \log n\, SP(n,m))$ idejű algoritmust fejlesztett ki. Ezután Orlin 1988-ban [54] egy $O(m \log n\, SP(n,m)) = O(m \log n(m + n \log n))$ kapacitásskálázó algoritmust adott, amely jelenleg a legjobb erősen polinomiális aszimptotikus korlátot biztosítja. Ezek az algoritmusok többnyire az előző alfejezetben tárgyalt módszerek módosított, továbbfejlesztett változatai voltak, amelyek a skálázáson kívül felhasználtak egyéb technikákat is. Általában a közbenső lépésekben felismerik, hogy az optimális megoldás bizonyos részeit már megtalálták, és ezeket rögzítve csökkentik a probléma méretét.

Hasonló elv érvényesül a *minimális átlagú negatív kör* (*minimum mean cycle-canceling*) algoritmusban is, amelyet 1988-ban Goldberg és Tarjan [33] mutatott be. Ez az algoritmus a 2.2.1. szakaszban tárgyalt negatív kör módszer $O(n^2 m^3 \log n)$ idejű speciális változata, amely minden iterációban egy minimális átlagköltségű kör mentén javít. Jelentőségét elsősorban az adja, hogy a többi erősen polinomiális algoritmusnál lényegesen egyszerűbb, viszont messze nem olyan hatékony. A 3.2. alfejezetben ennek az algoritmusnak a megvalósítását is bemutatjuk.

2.5. Legjobb lépésszámkorlátok

A fejezetben felsorolt eredményeket összegezve az alábbi táblázatban megadjuk a minimális költségű folyam feladatra a legjobb lépésszámkorlátokat biztosító algoritmusokat, elsősorban a [2, 60] könyvek alapján.

$O(nU\, SP(n, m, nC))$	Edmonds és Karp (1970) [21], Tomizawa (1971) *ismételt legrövidebb út*
$O(m \log U\, SP(n, m, nC))$	Edmonds és Karp (1972) [21] *kapacitásskálázás*
$O(n^{5/3} m^{2/3} \log(nC))$	Goldberg és Tarjan (1987) [32] *költségskálázás*
$O(m \log n\, SP(n, m))$	Orlin (1988) [54] *továbbfejlesztett kapacitásskálázás*
$O(nm \log \log U \log(nC))$	Ahuja, Goldberg, Orlin és Tarjan (1988) [1] *kétszeres skálázás*
$O(nm \log(n^2/m) \log(nC))$	Goldberg és Tarjan (1990) [34] *költségskálázás*

1. táblázat. A legjobb lépésszámkorlátokat adó algoritmusok

3. Megvalósítás

Ebben a fejezetben az általunk megvalósított minimális költségű folyam-algoritmusok elvét és főbb lépéseit ismertetjük, valamint megadunk ezeket szemléltető pszeudokódokat is. A fejlesztés célja az volt, hogy többféle módszert minél hatékonyabban implementáljunk, és ezzel széles körben felhasználható eszközöket dolgozzunk ki. Három alapvetően különböző megközelítési módot alkalmazva öt algoritmust valósítottunk meg: a negatív kör (primál) módszer egy egyszerűbb és egy erősen polinomiális változatát, az ismételt legrövidebb út (duál) algoritmust és annak kapacitásskálázó változatát, valamint a hálózati szimplex algoritmust. A negatív kör módszerek megvalósítását elsősorban az egyszerűségük és az erősen polinomiális változat elméleti jelentősége, a duál algoritmusok és a hálózati szimplex algoritmus implementálását pedig a kiemelkedő hatékonyságuk motiválta.

3.1. Az implementáció kerete

A fejezetben bemutatásra kerülő algoritmusokat az Eötvös Loránd Tudományegyetemen fejlesztett LEMON könyvtár keretében implementáltuk. A LEMON (*Library of Efficient Models and Optimization in Networks*) egy gráfelméleti, hálózattervezési és egyéb kombinatorikus optimalizálási problémák megoldására készített C++ könyvtár, amely forráskóddal együtt szabadon hozzáférhető. A programcsomag alapjait a gráfszerkezetek és a gráfokhoz kapcsolódó adatok kezelésére kifejlesztett adatstruktúrák, valamint a hatékonyan megvalósított gráfelméleti és optimalizálási algoritmusok adják. Elsődleges tervezési szempont volt, hogy különböző problémák megoldása során ezeket az algoritmusokat minél hatékonyabban, ugyanakkor egyszerűen lehessen alkalmazni, ezért a könyvtár a modern programfejlesztési elveket követve generikus programozási módszertan szerint készült, a C++ Standard Template Library eszközeinek felhasználásával. A LEMON könyvtárról részletesebben a `http://lemon.cs.elte.hu` címen tájékozódhatunk.

A minimális költségű folyam-algoritmusok implementálása során a LEMON által biztosított eszközök közül elsősorban a gráfok kezelésére kifejlesztett adatszerkezeteket, valamint a legrövidebb út és a maximális folyam problémára adott ismert algoritmusokat használtunk fel: a Dijkstra-algoritmust, a Bellman–Ford-algoritmust és az előfolyam algoritmust [15, 2].

3.2. Negatív kör algoritmusok

Az 1.10. optimalitási feltételből közvetlenül adódik a negatív kör módszer, amelyet a 2.2.1. szakaszban röviden már bemutattunk. Először meghatározunk egy megengedett folyamot, majd minden iterációban ennek célfüggvényértékén próbálunk javítani. Ha a folyamhoz tartozó maradék hálózatban találunk negatív kört, akkor azon a legkisebb

maradék kapacitású élének megfelelő mennyiségű folyamot küldünk, és ezáltal kiiktatjuk a maradék hálózatból, ha pedig már nem találunk, akkor az említett feltétel értelmében a folyam optimális. A 4. ábrán megadjuk a módszer általános változatát, r_{ij}-vel jelölve a maradék hálózat (i, j) élének maradék kapacitását.

procedure NEGATÍV KÖR MÓDSZER
egy x megengedett folyam meghatározása
while G_x tartalmaz negatív kört **do**
egy W negatív kör meghatározása
$\delta \leftarrow \min\{r_{ij} : (i, j) \in W\}$
δ egységnyi folyam küldése a W kör mentén, x és G_x frissítése
end while
end procedure

4. ábra. Negatív kör módszer

Mivel feltettük, hogy minden adat egész, az algoritmus véges sok lépésben biztosan megtalál egy optimális megoldást (amennyiben létezik megengedett megoldás). Ebből pedig következik az alábbi tétel, amelyre már korábban utaltunk.

3.1. Tétel. *Ha a minimális költségű folyam feladatban minden termelés/fogyasztás érték és kapacitás egész, és létezik megengedett megoldás, akkor létezik egészértékű optimális megoldás is.*

Bizonyítás. A tételt a negatív kör módszer iterációinak számára vonatkozó indukcióval bizonyíthatjuk. Egész kapacitások esetén a maximális folyam feladatnak létezik egészértékű megoldása [15, 2], így az algoritmus első lépésében a 3.2.1. szakaszban bemutatott módon meghatározhatunk egy egészértékű megengedett folyamot. Ha pedig egy iteráció kezdetén az aktuális folyam egészértékű, akkor minden maradék kapacitás egész, tehát a talált kör mentén történt javítással is ilyen folyamot kapunk. Így az algoritmus minden lépésében egészértékű folyamot tartunk nyilván, tehát a talált optimális megoldás is egészértékű lesz. ■

3.2.1. Megengedett folyam keresése

Az alábbiakban megmutatjuk, hogy a minimális költségű folyam feladat egy megengedett megoldásának keresése visszavezethető egy maximális folyam feladatra. Ehhez egészítsük ki a gráfot egy s forrás csúccsal és egy t nyelő csúccsal oly módon, hogy minden i termelő csúcshoz felveszünk egy (s, i) élt $b(i)$ kapacitással, és minden i fogyasztó csúcshoz egy (i, t) élt $-b(i)$ kapacitással. Könnyű látni, hogy ha ebben a gráfban keressünk egy maximális folyamot s-ből t-be, akkor arra teljesül a következő állítás.

3.2. Tétel. *Az eredeti feladatnak akkor és csak akkor létezik megengedett megoldása, ha a segédgráfban talált maximális folyam telíti az összes s-ből induló, illetve az összes t-be vezető élt, továbbá egy ilyen folyam az eredeti gráf élein egy megengedett megoldást ad.*

Vegyük észre, hogy mivel $\sum_{i \in V} b(i) = 0$, az s-ből induló élek pontosan akkor lesznek telítve, amikor a t-be vezető élek, tehát ez a két feltétel ekvivalens.

Bizonyítás. Ha x az eredeti feladat egy megengedett megoldása, akkor minden (s, i) élre $x_{si} = b(i)$ és minden (i, t) élre $x_{it} = -b(i)$ választással az eredeti gráf összes csúcsában teljesül a folyammegmaradás feltétele, és mivel s minden kimenő, illetve t minden bemenő kapacitása le van kötve, a folyam maximális. Másrészt, ha a segédgráfban egy x maximális folyam telíti az összes s-ből induló, illetve az összes t-be vezető élt, akkor a folyamot az eredeti gráf éleire megszorítva minden csúcsban teljesül a termelés/fogyasztás feltétel, tehát egy megengedett megoldást kapunk. Mivel minden kapacitás egész, feltehetjük, hogy az így talált folyam is egészértékű. ■

A LEMON-ban viszont szerepel egy, az előfolyam módszerhez hasonló algoritmus, amely közvetlenül alkalmazható a megengedett folyam-keresésnél általánosabb feladatra, melyben a termelés/fogyasztás feltételekkel egyenlőség helyett kisebb vagy egyenlő korlátozásokat adunk meg. Ezt alkalmazva elkerülhető a fenti transzformáció, vagyis a gráf lemásolása és kiterjesztése, amely nagyságrendileg annyi időt igényel, mint egy maximális folyam-keresés az előfolyam algoritmus felhasználásával. Így általában hatékonyabb implementációt kaptunk (l. 4.2.1. szakasz), ezért ezt a módszert választottuk.

3.2.2. Egyszerű negatív kör algoritmus

Az általános negatív kör módszerben nincs meghatározva, hogy a negatív köröket milyen algoritmussal keressük meg, illetve milyen sorrendben válasszuk ki. Az egyes változatok hatékonysága azonban elméleti és gyakorlati szempontból is nagyon különböző lehet.

A [14] cikk számos módszert bemutat negatív körök keresésére, amelyek közül az egyik legegyszerűbb a Bellman–Ford-algoritmuson alapul. Ez az algoritmus egy csúcsból induló legkisebb költségű (legrövidebb) utakat keres egy gráfban, amelyben szerepelhetnek negatív élköltségek is, de nincs negatív kör, azonban alkalmas az utóbbi feltétel ellenőrzésére is [15, 2]. Ismert, hogy az algoritmus $n - 1$ iterációja után egy n-edik iterációt is végrehajtva, abban akkor és csak akkor javítjuk valamely csúcs távolságcímkéjét, ha a gráf tartalmaz a kezdőcsúcsból elérhető negatív kört. Ha ugyanis történt ilyen javítás, akkor azt egy legalább n hosszú séta eredményezte – hiszen az első $n - 1$ iterációban még nem találtuk meg –, amelynek költsége minden legfeljebb $n - 1$ hosszú séta költségénél kisebb, tehát ebben szerepelnie kell egy negatív összköltségű körnek. Az algoritmus működéséből adódik, hogy ha ilyenkor az utolsó iterációban javított csúcsból elindulva

a szülőmutatók mentén visszafelé lépkedünk, akkor előbb-utóbb olyan csúcsba jutunk, ahol már jártunk, és az így kapott kör szükségszerűen egy negatív kör lesz.

Ezt a módszert alkalmazva egy egyszerű negatív kör algoritmus adódik: minden lépésben végrehajtjuk a Bellman–Ford-algoritmust, ami $O(nm)$ időt igényel, majd a szülőmutatók segítségével meghatározunk egy negatív kört (ha ilyen volt), és ezen javítunk, ami $O(n)$ lépésben megtehető. Mivel a kezdeti megoldás összköltségének mCU felső korlátja, és ezt minden iterációban legalább eggyel javítjuk, az algoritmus $O(nm^2CU)$ időben fut.

A gyakorlatban azonban általában gyorsabban is találhatunk negatív köröket. A Bellman–Ford-algoritmus szokásos megvalósítása szerint az egyes iterációkban felhasználjuk azokat az eredményeket is, amelyeket az adott iterációban már megkaptunk, így az algoritmus „előreszaladhat”, vagyis a k-adik iterációban megtalálhat k-nál hosszabb sétákat is. Ez pedig azt sugallja, hogy egy negatív kör megtalálásához általában n-nél jóval kevesebb iteráció is elég lehet, amit számos mérési eredménnyel alá is tudtunk támasztani. A nehézséget az okozza, hogy a Bellman–Ford-algoritmus futása során nem tudjuk hatékonyan ellenőrizni, hogy találtunk-e már negatív kört, ezért érdemes időnként megszakítani, és elvégezni ezt az ellenőrzést.

Ezek alapján a fentinél hatékonyabb algoritmust kaphatunk, amelyet az 5. ábrán mutatunk be. Jelöljön $S \geq 1$ és $\alpha > 1$ egy-egy rögzített paramétert, és kezdetben legyen $K = S$. Minden lépésben először csak K Bellman–Ford-iterációt végzünk, és ha ezzel nem találunk negatív kört, akkor legyen $K = \alpha K$, majd végrehajtunk újabb K iterációt. (Vegyük észre, hogy ilyenkor nem szükséges elölről kezdenünk a Bellman–Ford-algoritmust, folytathatjuk onnan, ameddig eljutottunk.) Így az algoritmus működése során eleinte kevés iterációval is találhatunk negatív köröket, és csak akkor növeljük a K korlátot, amikor erre már szükség van. Ezen kívül további javítást érhetünk el azáltal, ha egy lépésben nem csak egy negatív kört szüntetünk meg. Vegyük észre ugyanis, hogy a Bellman–Ford-algoritmus futása során a szülőmutatók által meghatározott gráfban egyszerre több csúcsdiszjunkt negatív kör is megjelenhet, amelyek $O(n)$ időben megtalálhatók.

Megjegyezzük továbbá, hogy a Bellman–Ford-algoritmusban érdemes minden i csúcsot kezdőcsúcsnak tekinteni $d(i) = 0$ távolsággal, ugyanis így egy kereséssel elérhető lesz az összes csúcs. Ez annak felel meg, mintha az aktuális folyamhoz tartozó maradék hálózatot kiegészítenénk egy olyan forrás csúccsal, amelyből minden más csúcsba egy nulla költségű él vezet, és ebből a csúcsból indítanánk a keresést.

Az algoritmus az általános módszerhez hasonlóan $O(nm^2CU)$ időben fut, ennél jobb korlátot általában nem tudunk adni, viszont a gyakorlatban sokkal hatékonyabbnak bizonyult. Természetesen az S és az α paraméter értéke alapvetően befolyásolja a futásidőt, ezért ezeknek számos kombinációját kipróbáltuk. A mérési tesztjeink alapján $S = 2$, $\alpha = \frac{3}{2}$ bizonyult a legjobb választásnak (l. 4.2.2. szakasz).

procedure EGYSZERŰ NEGATÍV KÖR ALGORITMUS
egy x megengedett folyam meghatározása
$K \leftarrow S$
$opt \leftarrow hamis$ ▷ *optimális-e a megoldás*
while $\neg\, opt$ **do**
BF-algoritmus inicializálása: $\forall i \in V : d(i) \leftarrow 0$
$k \leftarrow 0$ ▷ *végrehajtott BF-iterációk száma*
$c \leftarrow hamis$ ▷ *találtunk-e negatív kört*
while $\neg\, c$ **do**
legfeljebb $\min\{K, n-k\}$ BF-iteráció végrehajtása, k frissítése
if egy iterációban nem történt javítás **then**
$opt \leftarrow igaz$
kilépés a ciklusból
end if
negatív kör keresése a $p(\cdot)$ szülőmutatók alapján
while találtunk egy W negatív kört **do**
$c \leftarrow igaz$
$\delta \leftarrow \min\{r_{ij} : (i,j) \in W\}$
δ egységnyi folyam küldése a W kör mentén, x és G_x frissítése
negatív kör keresése a $p(\cdot)$ szülőmutatók alapján
end while
if $\neg\, c$ **then**
$K \leftarrow \lfloor \alpha K \rfloor$ ▷ *korlát növelése*
end if
end while
end while
end procedure

5. ábra. Egyszerű negatív kör algoritmus

3.2.3. Minimális átlagú negatív kör algoritmus

Az előző szakaszban tárgyalt algoritmuson kívül a negatív kör módszer egy erősen polinomiális változatát, a *minimális átlagú negatív kör* algoritmust [33] is megvalósítottuk, amelyet a 6. ábrán mutatunk be. Ez az algoritmus minden iterációban egy minimális átlagköltségű negatív kör mentén javít, tehát egy olyan W kör mentén, amelyre a $(\sum_{(i,j)\in W} c_{ij})/|W|$ érték minimális. Belátható, hogy ezzel a választási szabállyal a negatív kör módszer erősen polinomiális időben fut.

3.3. Tétel. *A minimális átlagú negatív kör algoritmus tetszőleges valós élköltségek esetén* $O(nm^2 \log n)$ *iterációt végez, tehát* $O(n^2 m^3 \log n)$ *időben fut, egész élköltségek esetén pedig* $O(nm \log(nC))$ *iterációt végez, tehát* $O(n^2 m^2 \min\{\log(nC), m \log n\})$ *időben fut.*

procedure MINIMÁLIS ÁTLAGÚ NEGATÍV KÖR ALGORITMUS
egy x megengedett folyam meghatározása
egy minimális átlagköltségű W kör meghatározása
while W átlagköltsége negatív **do**
$\delta \leftarrow \min\{r_{ij} : (i,j) \in W\}$
δ egységnyi folyam küldése a W kör mentén, x és G_x frissítése
egy minimális átlagköltségű W kör meghatározása
end while
end procedure

6. ábra. Minimális átlagú negatív kör algoritmus

Annak ellenére, hogy az algoritmusban semmilyen skálázást nem alkalmazunk, a fenti tétel a költségskálázáshoz kapcsolódó ϵ-optimalitás fogalmát felhasználva igazolható. A részletes bizonyítás megtalálható Goldberg és Tarjan [33] cikkében, valamint Ahuja, Magnanti és Orlin [2] könyvében, mivel azonban meglehetősen hosszú, jelen dolgozat keretében nem ismertetjük.

3.2.4. Minimális átlagú kör keresése

A minimális átlagú negatív kör algoritmus megvalósításához természetesen szükségünk van arra, hogy megkeressünk egy minimális átlagköltségű kört. Erre a feladatra Karp 1978-ban [44] egy $O(nm)$ idejű algoritmust adott, amely a 3.4. tételen alapul. Mivel egy irányított kör csak egy erősen összefüggő komponensben fordulhat elő, a gráfot először bontsuk fel ilyen komponensekre, ami két mélységi keresés felhasználásával $O(n+m)$ időben elvégezhető. Feltehetjük tehát, hogy a vizsgált gráf erősen összefüggő, így egy tetszőlegesen rögzített s csúcsból minden más csúcs elérhető. Jelölje $\delta_k(i)$ a pontosan k élből álló legrövidebb $s \rightsquigarrow i$ séta költségét, illetve legyen $\delta_k(i) = \infty$, ha ilyen séta nem létezik. Ekkor teljesül az alábbi tétel [15, 44].

3.4. Tétel (Karp tétele). *A gráfban szereplő irányított körök átlagköltségének minimuma:*

$$\lambda^* = \min_{i \in V} \max_{0 \le k \le n-1} \frac{\delta_n(i) - \delta_k(i)}{n-k}.$$

Ebből a tételből kiindulva könnyen adható egy dinamikus programozási algoritmus a minimális körátlag kiszámítására. A 7. ábrán Karp eredeti algoritmusának általunk megvalósított változatát mutatjuk be, amely egy minimális átlagú kört is meghatároz. A $\delta_k(i)$ értékek mellett nyilvántartunk $p_k(i)$ szülőmutatókat is, amelyek segítségével az algoritmus által adott i^* csúcsból kiindulva megkaphatunk egy minimális átlagú kört.

Erre a feladatra számos más megoldási módszer is ismert [18, 19], amelyek többnyire Karp algoritmusának továbbfejlesztett, különböző heurisztikákkal javított változatai,

illetve közelítő algoritmusok. Ezeket azonban nem valósítottuk meg, ugyanis a minimális átlagú negatív kör algoritmus – elméleti jelentősége ellenére – a gyakorlatban összehasonlíthatatlanul lassabbnak bizonyult az összes többi módszernél, amelyet implementáltunk, beleértve a 3.2.2. szakaszban bemutatott negatív kör algoritmust is.

```
procedure KARP-ALGORITMUS
    for k ← 0 to n do                                   ▷ inicializálás
        for all i ∈ V do
            δ_k(i) ← ∞                  ▷ a k élből álló legrövidebb s ⇝ i séta hossza
            p_k(i) ← NIL                    ▷ az i-t megelőző csúcs ebben a sétában
        end for
    end for
    δ_0(s) ← 0
    for k ← 1 to n do                          ▷ δ_k(i) és p_k(i) értékek meghatározása
        for all (i, j) ∈ E do
            if δ_{k−1}(i) + c_ij < δ_k(j) then
                δ_k(j) ← δ_{k−1}(i) + c_ij
                p_k(j) ← i
            end if
        end for
    end for
    λ* ← ∞                                              ▷ a minimális körátlag
    for all i ∈ V do
        λ ← −∞                                ▷ a minimális körátlag az s ⇝ i sétákban
        for k ← 1 to n do
            λ ← max{λ, (δ_n(i) − δ_k(i))/(n − k)}
        end for
        if λ < λ* then
            λ* ← λ                          ▷ az átlagköltsége egy minimális átlagú körnek
            i* ← i                  ▷ egy ilyen kör szerepel az n élből álló s ⇝ i* sétában
        end if
    end for
end procedure
```

7. ábra. Karp-algoritmus

3.3. Duál algoritmusok

Ebben az alfejezetben bemutatjuk az általunk megvalósított két duál módszert: az ismételt legrövidebb út algoritmust, valamint ennek kapacitásskálázó változatát. Mindkettő a legjobb lépésszámkorlátokat adó algoritmusok közé tartozik, és a gyakorlatban is igen hatékonyak.

3.3.1. Ismételt legrövidebb út algoritmus

Az ismételt legrövidebb út módszert a 2.2.2. szakaszban röviden már bemutattuk. Ebben mindvégig olyan folyamot tartunk nyilván, amelyre teljesülnek a nemnegativitási és kapacitásfeltételek, viszont megsértheti a termelés/fogyasztás feltételeket, valamint olyan csúcspotenciálokat, amelyekkel erre a folyamra teljesül a redukált költség optimalitási feltétel. Ez kezdetben azonosan nulla folyammal és azonosan nulla potenciálfüggvénnyel könnyen biztosítható. Ezután minden lépésben kiválasztunk egy csúcsot, ahol még többlet van, és ebből folyamot küldünk egy hiánnyal rendelkező csúcsba a maradék hálózat egy, a redukált költségek szerint vett legrövidebb útja mentén. Tehát arra törekszünk, hogy végül teljesüljenek a termelés/fogyasztás feltételek is, vagyis megengedett folyamot kapjunk, amely a redukált költség feltétel megtartása miatt nyilván optimális lesz.

Az algoritmus tárgyalásához először bevezetünk még egy jelölést, majd kimondunk egy fontos segédállítást. Legyen a továbbiakban x egy olyan (nem feltétlenül megengedett) folyam, amely teljesíti a nemnegativitási és kapacitásfeltételeket, valamint egy adott π potenciálfüggvénnyel az 1.11. redukált költség optimalitási feltételt. Jelölje $e(i)$ azt az előjeles mennyiséget, amennyivel az i csúcsban a kifolyó és befolyó folyam különbsége eltér a $b(i)$ termelés/fogyasztás értéktől:

$$e(i) = b(i) - \sum_{j\,:\,(i,j)\in E} x_{ij} + \sum_{j\,:\,(j,i)\in E} x_{ji}.$$

Ha $e(i) > 0$, akkor *többletnek*, ha $e(i) < 0$, akkor *hiánynak* nevezzük.

3.5. Állítás. *A d függvény jelölje a G_x maradék hálózat csúcsainak egy rögzített s kezdőcsúcstól mért távolságát a c^{π}_{ij} redukált költségek szerint. Legyen továbbá x' egy olyan folyam, amelyet az x-ből úgy kapunk, hogy egy adott folyammennyiséget átküldünk az s csúcsból valamely másik csúcsba egy legrövidebb út mentén. Ekkor teljesülnek az alábbiak.*

a) Az x folyam a $\pi' = \pi - d$ potenciállal is teljesíti a redukált költség optimalitási feltételt.

b) $c^{\pi'}_{ij} = 0$ minden (i,j) élre, amely az s-ből valamely másik csúcsba vezető legrövidebb úton szerepel.

c) Az x' folyam a π' potenciállal teljesíti a redukált költség optimalitási feltételt.

Bizonyítás. Feltettük, hogy x-re és π-re teljesül a redukált költség optimalitási feltétel, tehát a G_x maradék hálózat minden (i,j) élére $c^{\pi}_{ij} \geq 0$, továbbá a d távolságfüggvényre teljesül, hogy $d(j) \leq d(i) + c^{\pi}_{ij}$ a G_x minden (i,j) élére. Az utóbbi feltételbe behelyettesítve a redukált költség definícióját, azt kapjuk, hogy $d(j) \leq d(i) + c_{ij} - \pi(i) + \pi(j)$, vagyis:

$$c^{\pi'}_{ij} = c_{ij} - (\pi(i) - d(i)) + (\pi(j) - d(j)) \geq 0.$$

Ezzel az állítás *a)* pontját beláttuk.

Egy legrövidebb úton szereplő (i, j) élre $d(j) = d(i) + c_{ij}^{\pi}$, ebből pedig c_{ij}^{π} definícióját behelyettesítve az előzőekhez hasonlóan kapjuk, hogy $c_{ij}^{\pi'} = 0$, így beláttuk a *b)* összefüggést is.

Mivel az *a)* pont szerint az x folyamhoz a π' is megfelelő potenciál, az utolsó pont bizonyításához csak azokat az éleket kell megvizsgálnunk, amelyeken a $G_{x'}$ maradék hálózat különbözhet a G_x-től. Legyen (i, j) egy tetszőleges éle az adott legrövidebb útnak, amelyen folyamot küldünk. A változtatás által egyrészt az (i, j) él eltűnhet a maradék hálózatból, másrészt belép a (j, i) él (ha eddig nem szerepelt). Az előbbi természetesen nem okoz gondot, és mivel a *b)* pont alapján $c_{ij}^{\pi'} = 0$, ezért $c_{ji}^{\pi'}$ is nulla lesz, tehát minden belépő élre is teljesülni fog a redukált költség optimalitási feltétel. ■

Ezek után a 8. ábrán látható módon megfogalmazhatjuk az ismételt legrövidebb út algoritmust. A fenti állítás alapján minden lépésben egy többlettel rendelkező csúcsból kiindulva legrövidebb út-keresést végzünk, majd a kapott távolságoknak megfelelően módosítjuk a csúcspotenciálokat, és folyamot küldünk egy legrövidebb út mentén egy hiánnyal rendelkező csúcsba. A potenciálok két szempontból is kulcsfontosságú szerepet töltenek be: egyrészt igazolják az algoritmus helyességét, vagyis a talált folyam optimalitását, másrészt biztosítják a nemnegatív élköltségeket a legrövidebb út-keresésekhez, így ezeket Dijkstra algoritmusával hatékonyan megvalósíthatjuk.

procedure ISMÉTELT LEGRÖVIDEBB ÚT ALGORITMUS
  $x \leftarrow 0$, $\pi \leftarrow 0$
  $e(i) \leftarrow b(i)$ minden $i \in V$ csúcsra
  $S \leftarrow \{i \in V : e(i) > 0\}$ ▷ *többlettel rendelkező csúcsok*
  $T \leftarrow \{i \in V : e(i) < 0\}$ ▷ *hiánnyal rendelkező csúcsok*
  while $S \neq \emptyset$ **do**
    Dijkstra-algoritmus futtatása egy $v \in S$ csúcsból indulva a G_x hálózatban
      a c_{ij}^{π} redukált élköltségek szerint
    if nem értünk el T-beli csúcsot **then**
      kilépés, nem létezik megengedett folyam
    end if
    legyen P egy legrövidebb út v-ből egy $w \in T$ csúcsba
    $\pi \leftarrow \pi - d$
    $\delta \leftarrow \min\{e(v), -e(w), \min\{r_{ij} : (i, j) \in P\}\}$
    δ egységnyi folyam küldése a P út mentén
    x, e, G_x, S, T és a redukált költségek frissítése
  end while
end procedure

8. ábra. Ismételt legrövidebb út algoritmus

Az első lépésben azonosan nulla x és π mellett nyilván teljesül a redukált költség optimalitási feltétel, és a 3.5. állítás alapján ezt minden iterációban fenntartjuk. Ha már nincsen többlettel rendelkező csúcs, akkor az algoritmus véget ér, az eredmény pedig biztosan egy megengedett optimális folyam lesz. (Vegyük észre, hogy $\sum_{i \in V} b(i) = 0$ miatt az $S \neq \emptyset$ és a $T \neq \emptyset$ feltétel ekvivalens.) Ezen kívül azt kell még belátnunk, hogy ha létezik megengedett megoldás, akkor az algoritmus nem akad el, vagyis minden lépésben találunk megfelelő utat, amely mentén javíthatunk. Ezt mondja ki az alábbi tétel.

3.6. Tétel. *A minimális költségű folyam feladatnak akkor és csak akkor létezik megengedett megoldása, ha az ismételt legrövidebb út algoritmus minden lépésében bármely $v \in S$ csúcsból elérhető valamely $w \in T$ csúcs.*

Bizonyítás. A tétel egyik fele nyilvánvaló: vegyük észre, hogy az algoritmus futása során az S és a T halmaz nem bővülhet, így ha minden lépésben találunk megfelelő utat, akkor az algoritmus véges sok lépésben egy megengedett folyamot eredményez. A másik irány igazolásához vegyük alapul a 3.2.1. szakaszban adott struktúrát. Tegyük fel, hogy létezik megengedett megoldás, és legyen x az algoritmus egy köztes lépésében nyilvántartott folyam. Egészítsük ki a G_x maradék hálózatot egy s forrás csúccsal és egy t nyelő csúccsal oly módon, hogy minden i termelő csúcshoz felveszünk egy (s, i) élt, és minden i fogyasztó csúcshoz egy (i, t) élt. Egyszerű meggondolással adódik, hogy az (s, i) éleken $u_{si} = b(i)$, $x_{si} = b(i) - e(i)$, az (i, t) éleken pedig $u_{it} = -b(i)$, $x_{it} = e(i) - b(i)$ választással minden valódi csúcsban egyenlő lesz a kifolyó és befolyó folyam nagysága, vagyis egy s–t folyamot kapunk. Legyen $v \in S$ tetszőleges csúcs. Ekkor v biztosan termelő csúcs volt, tehát létezik (s, v) él, amely még nincs telítve. Mivel feltettük, hogy a feladatnak létezik megengedett megoldása, a 3.2. tétel alapján biztosan létezik olyan $s \rightsquigarrow t$ javítóút, amely áthalad a v csúcson, ugyanis a maximális folyam az (s, v) élt is telíti. Ez a javítóút pedig biztosan áthalad valamely $w \in T$ csúcson is, hiszen a T halmazban már nem szereplő fogyasztó csúcsokból a t csúcsba vezető élek már telítve vannak. Tehát ennek az útnak az eredeti maradék hálózatba eső része éppen egy megfelelő $v \rightsquigarrow w$ utat ad, és ezzel a tétel másik irányát is beláttuk. ■

A fenti tétellel tehát igazoltuk az algoritmus helyességét. Mivel $\sum_{i \in V} b(i) = 0$, és U a $|b(i)|$ értékek felső korlátja, kezdetben legfeljebb $nU/2$ lehet az összes többlet, amelyet minden iterációban csökkentünk, így az algoritmus $O(nU\, SP(n, m, nC))$ időben fut.

A megvalósítás során azonban alkalmaztunk még egy fontos javítást, amellyel az algoritmus lényegesen gyorsabbá tehető. Az egyes iterációk során nem szükséges minden csúcs távolságát meghatároznunk, a Dijkstra-algoritmust megszakíthatjuk akkor, amikor véglegesen címkéz egy hiánnyal rendelkező w csúcsot, vagyis amikor meghatároz egy oda vezető legrövidebb utat. Ezután a π potenciált a következőképpen módosíthatjuk:

$$\pi'(i) = \begin{cases} \pi(i) - d(i), & \text{ha az } i \text{ csúcsot már véglegesen címkéztük;} \\ \pi(i) - d(w) & \text{különben.} \end{cases}$$

A 3.5. állítás bizonyításához hasonlóan erre a π'-re is igazolhatjuk mindhárom szükséges feltételt. Vegyük észre továbbá, hogy egy potenciálfüggvényhez egy konstans értéket hozzáadva minden redukált költség változatlan marad, tehát a fenti módosítással egyenértékű a következő:

$$\pi'(i) = \begin{cases} \pi(i) - d(i) + d(w), & \text{ha az } i \text{ csúcsot már véglegesen címkéztük;} \\ \pi(i) & \text{különben.} \end{cases}$$

Így viszont csak azoknak a csúcsoknak kell módosítanunk a potenciálját, amelyeket a Dijkstra-algoritmus véglegesen címkéz, tehát az adott iterációban a többi csúccsal semmilyen műveletet nem kell végeznünk.

A 4. fejezetben ismertetett mérési eredmények azt mutatják, hogy ezzel a módosítással az algoritmus meglehetősen hatékony, különösen olyankor, amikor minden termelés/fogyasztás érték viszonylag kicsi.

3.3.2. Kapacitásskálázó algoritmus

Megvalósítottunk egy kapacitásskálázó algoritmust is, amely Edmonds és Karp eredeti módszerének [21] egy módosított változatán alapul. Az utóbbi algoritmus Orlintól származik [54], aki ebből kiindulva fejlesztette ki a legjobb ismert erősen polinomiális futási időt adó algoritmust.

Először Orlin algoritmusát tárgyaljuk, amelyről bizonyítjuk, hogy polinomiális időben fut, majd bemutatjuk az általunk módosított változatot is. Az előbbihez azonban szükségünk lesz egy további kikötésre: tegyük fel, hogy a G gráfban bármely két csúcs között vezet végtelen, illetve kellően nagy kapacitású irányított út. Ezt könnyen biztosíthatjuk oly módon, hogy valamely rögzített s csúcsra a gráf minden más i csúcsához felveszünk egy (s, i) és egy (i, s) élt kellően nagy kapacitással és élköltséggel. Egy optimális megoldás ezeket az éleket nyilván nem fogja használni, tehát az így kapott feladat ekvivalens az eredetivel.

Az ismételt legrövidebb út módszer alapvető hátránya az, hogy az egyes lépésekben olyan utakat találhatunk meg, amelyeken csak kis folyammennyiséget tudunk átküldeni. A vizsgált algoritmus ezt a jelenséget igyekszik kiküszöbölni kapacitásskálázó technika alkalmazásával. Minden iterációban a Δ paraméter aktuális értéke mellett olyan legrövidebb utakat keresünk, amelyeken legalább Δ egységnyi folyamot küldhetünk át. Amennyiben ilyen utat már nem találunk, felezzük Δ értékét, és végrehajtunk egy újabb fázist. Kezdetben $\Delta = 2^{\lfloor \log U \rfloor}$, és a $\Delta = 1$ fázis befejezésekor egy megengedett folyamot kapunk, amely egyben optimális is lesz, hiszen a redukált költség optimalitási feltételt itt is megtartjuk.

Az algoritmus tárgyalásához vezessünk be még néhány újabb jelölést. Legyen S_Δ a legalább Δ többlettel, T_Δ pedig a legalább Δ hiánnyal rendelkező csúcsok halmaza. Minden lépésben egy $v \in S_\Delta$ csúcs és egy $w \in T_\Delta$ csúcs között keresünk legrövidebb utat, és

azt szeretnénk, hogy ezen az úton az összes él maradék kapacitása is legalább Δ legyen. Ennek megfelelően vezessük be az adott x folyamhoz tartozó G_x^Δ Δ-*maradék hálózatot*, amely a G_x gráfnak a legalább Δ maradék kapacitású éleiből álló részgráfja. Az egyes iterációk során a G_x^Δ hálózatban keresünk legrövidebb utakat, és ebben a gráfban minden élre biztosítjuk a redukált költség optimalitási feltételt, a G_x hálózat Δ-nál kisebb maradék kapacitású éleire viszont nem. Ezen fogalmak felhasználásával az algoritmust a 9. ábrán mutatjuk be.

procedure KAPACITÁSSKÁLÁZÓ ALGORITMUS
  $x \leftarrow 0$, $\pi \leftarrow 0$
  $e(i) \leftarrow b(i)$ minden $i \in V$ csúcsra
  $\Delta \leftarrow 2^{\lfloor \log U \rfloor}$
  while $\Delta \geq 1$ **do**
    for a G_x^Δ minden (i,j) élére **do**
      if $c_{ij}^\pi < 0$ **then**
        r_{ij} egységnyi folyam küldése az (i,j) élen
        x, e, G_x^Δ frissítése
      end if
    end for
    $S_\Delta \leftarrow \{i \in V : e(i) \geq \Delta\}$
    $T_\Delta \leftarrow \{i \in V : e(i) \leq -\Delta\}$
    while $S_\Delta \neq \emptyset$ **and** $T_\Delta \neq \emptyset$ **do**
      egy $v \in S_\Delta$ és egy $w \in T_\Delta$ csúcs kiválasztása
      Dijkstra-algoritmus futtatása a v csúcsból indulva a G_x^Δ hálózatban
        a c_{ij}^π redukált élköltségek szerint
      legyen P a talált legrövidebb $v \rightsquigarrow w$ út a G_x^Δ hálózatban
      $\pi \leftarrow \pi - d$
      Δ egységnyi folyam küldése a P út mentén
      x, e, G_x^Δ, S_Δ, T_Δ és a redukált költségek frissítése
    end while
    $\Delta \leftarrow \Delta/2$
  end while
end procedure

9. ábra. Kapacitásskálázó algoritmus

Egy fázis akkor ér véget, ha S_Δ vagy T_Δ üres lesz, vagyis amikor a következő fázisba lépünk, az új Δ érték mellett $e(i) < 2\Delta$ minden i csúcsra vagy $e(i) > -2\Delta$ minden i csúcsra. Tehát a csúcsokban megjelenő összes többlet – amely megegyezik az összes hiánnyal – legfeljebb $2n\Delta$ lehet (ez nyilván igaz a legelső fázisban is). Minden fázisban a G_x^Δ hálózatba belépnek azok az élek, amelyek maradék kapacitása Δ és 2Δ között van. Ezekre nem feltétlenül teljesül a redukált költség optimalitási feltétel, ezért a fázis elején ezt ellenőrizzük. Ha valamely (i,j) élre a c_{ij}^π redukált költség negatív, akkor ezt az élt

telítjük, és ezzel kiiktatjuk a maradék hálózatból, az ezáltal belépő megfordított élre pedig $c^\pi_{ji} = -c^\pi_{ij}$ miatt teljesül az optimalitási feltétel. Ezen élek maradék kapacitása kisebb, mint 2Δ, tehát a telítések hatására az összes többlet növekedése kevesebb, mint $2m\Delta$. Összegezve tehát minden fázis elején $2(n+m)\Delta$ az összes többlet egy felső korlátja.

Minden lépésben egy $v \in S_\Delta$ csúcsból egy $w \in T_\Delta$ csúcsba küldünk folyamot. Mivel feltettük, hogy az eredeti gráfban bármely két csúcs között vezet kellően nagy kapacitású irányított út, a G^Δ_x hálózatban is biztosan találunk utat v és w között. Ezen az úton Δ egységnyi folyamot átküldve a v csúcsban Δ-val csökkentjük a többletet. A fázis elején viszont legfeljebb $2(n+m)\Delta$ az összes többlet, ezért legfeljebb $2(n+m)$ ilyen javítást végezhetünk. A fázisok száma pedig nyilván $O(\log U)$, tehát az algoritmus teljes műveletigénye $O(m \log U\, SP(n, m, nC))$.

A helyesség belátásához vegyük észre, hogy mivel minden adat egész, $\Delta = 1$ esetén $G^\Delta_x = G_x$, így az algoritmus végén a maradék hálózat minden élére biztosítjuk a redukált költség optimalitási feltételt. Tehát az eredményül kapott folyam pontosan akkor használ újonnan hozzávett nagy költségű élt, ha az eredeti feladatnak nem létezik megengedett megoldása.

A megvalósítás során azonban szerettük volna elkerülni a fent tárgyalt gráfátalakítást. Ehhez viszont módosítanunk kell az algoritmust is, hiszen ebben az esetben egy adott lépésben kiválasztott $v \in S_\Delta$ csúcsból kiindulva nem feltétlenül fogunk alkalmas utat találni. Ilyenkor megtehetjük azt, hogy ezt a csúcsot kivesszük S_Δ-ból, és csak a következő fázisban próbáljuk meg a többletét elküldeni. Ha azonban $\Delta = 1$ esetén is van ilyen csúcs, akkor a feladatnak nem létezik megengedett megoldása.

Az összehasonlítás érdekében a transzformáció alkalmazásával megvalósítottuk az eredeti algoritmust is. A két változat minden tesztadatra közel azonos mennyiségű legrövidebb út-keresést végzett, viszont az eredeti algoritmus átlagosan 20 %-kal lassabbnak bizonyult, a gráfátalakítás költségét nem számítva (l. 4.2.5. szakasz).

Ezen a javításon kívül egyéb módosításokat is végeztünk. Itt is alkalmaztuk az előző szakaszban már tárgyalt technikát, miszerint az egyes lépésekben nem határozzuk meg az összes csúcsba vezető legrövidebb utat. Amikor az első $w \in T_\Delta$ csúcsot véglegesen címkézzük, akkor a Dijkstra-algoritmus futását megszakítjuk, és a potenciálokat az előző szakaszban megadott módon változtatjuk meg. A gyakorlatban az is jelentős javításnak bizonyult, ha a megtalált legrövidebb utakon Δ egységnyi folyam helyett minden esetben a lehető legnagyobb folyammennyiséget küldjük át (amely nyilván Δ és 2Δ között van). Vegyük észre továbbá, hogy az algoritmus helyességét nem befolyásolja, hogy a Δ értékét az egyes fázisokban milyen módon csökkentjük. Megtehetjük tehát, hogy nem felezzük, hanem egy tetszőlegesen rögzített $K > 1$ egésszel osztjuk el.

Ezekkel a módosításokkal a 10. ábrán megadott algoritmust kapjuk, amelynek helyességét a 3.6. tétel bizonyításához hasonlóan igazolhatjuk.

procedure MÓDOSÍTOTT KAPACITÁSSKÁLÁZÓ ALGORITMUS
$x \leftarrow 0$, $\pi \leftarrow 0$
$e(i) \leftarrow b(i)$ minden $i \in V$ csúcsra
$\Delta \leftarrow K^{\lfloor \log_K U \rfloor}$
while $\Delta \geq 1$ **do**
for a G_x^Δ minden (i,j) élére **do**
if $c_{ij}^\pi < 0$ **then**
r_{ij} egységnyi folyam küldése az (i,j) élen
x, e, G_x^Δ frissítése
end if
end for
$S_\Delta \leftarrow \{i \in V : e(i) \geq \Delta\}$
$T_\Delta \leftarrow \{i \in V : e(i) \leq -\Delta\}$
while $S_\Delta \neq \emptyset$ **and** $T_\Delta \neq \emptyset$ **do**
egy $v \in S_\Delta$ csúcs kiválasztása
Dijkstra-algoritmus futtatása a v csúcsból indulva a G_x^Δ hálózatban
a c_{ij}^π redukált élköltségek szerint
(az első T_Δ-beli csúcs végleges címkézésekor befejezzük a keresést)
if nem értünk el T_Δ-beli csúcsot **then**
if $\Delta > 1$ **then**
$S_\Delta \leftarrow S_\Delta \setminus \{v\}$
ugrás a következő iterációba
else
kilépés, nem létezik megengedett folyam
end if
end if
legyen P egy legrövidebb út v-ből egy $w \in T_\Delta$ csúcsba
for all $i \in V$, amelyet a Dijkstra-algoritmus véglegesen címkézett **do**
$\pi(i) \leftarrow \pi(i) - d(i) + d(w)$
end for
$\delta \leftarrow \min\{e(v), -e(w), \min\{r_{ij} : (i,j) \in P\}\}$
δ egységnyi folyam küldése a P út mentén
x, e, G_x^Δ, S_Δ, T_Δ és a redukált költségek frissítése
end while
$\Delta \leftarrow \Delta / K$
end while
end procedure

10. ábra. Módosított kapacitásskálázó algoritmus

3.4. Hálózati szimplex algoritmus

Az előző két alfejezetben bemutatott algoritmusokon kívül megvalósítottuk a primál hálózati szimplex algoritmust is, amely a mérési tesztjeink alapján egyértelműen a leghatékonyabbnak bizonyult. Ennek a módszernek kiterjedt irodalma van, melyből elsősorban Ahuja, Magnanti és Orlin [2] könyvét, valamint Kelly és O'Neill [45] dolgozatát vettük alapul. Az utóbbi online elérhető, és az algoritmus elméleti elemzésén túl pszeudokódok formájában bemutatja egy hatékony megvalósítás részleteit is, amelyhez képest nem alkalmaztunk lényeges módosításokat. Ezért hivatkozva ezen forrásokra az alábbiakban mindössze egy áttekintést adunk a hálózati szimplex módszerről, és csak a különböző pivotálási szabályokra térünk ki részletesebben, melyek az implementáció meghatározó részét képezik.

A hálózati szimplex módszer a korlátozott változókat kezelő általános szimplex módszer egy módosított változata, amely kihasználja a minimális költségű folyam feladat specialitásait. Az egyes változóknak a gráf élei felelnek meg, a bázismegoldásoknak pedig olyan feszítőfák, amelyek élein a folyamérték tetszőleges lehet, de a többi élen nullával vagy az él kapacitásával egyezik meg. Belátható, hogy ha egy minimális költségű folyam feladatnak létezik optimális megoldása, akkor létezik optimális bázismegoldás is.

Az algoritmusban tehát mindvégig egy feszítőfát, annak élein megfelelő folyamértékeket (primál megoldás), valamint csúcspotenciálokat (duál megoldás) tartunk nyilván, és az egyes iterációk során az aktuális célfüggvényértéken igyekszünk javítani. A redukált élköltségek alapján kiválasztunk egy fán kívüli élt, amely megsérti az 1.12. complementary slackness optimalitási feltételt, és hozzávesszük a feszítőfához (a bázishoz). Ezután az így kialakult kör mentén javítunk, és egy legkisebb maradék kapacitású élét kivesszük a bázisból. Ezt az élcserét és a feszítőfa-szerkezet frissítését együtt *pivotálásnak* nevezzük. Ha a gráfban már nincs alkalmas belépő él, akkor a talált megoldás optimális.

procedure HÁLÓZATI SZIMPLEX MÓDSZER
    egy megengedett bázismegoldást adó feszítőfa-szerkezet meghatározása
    while létezik olyan fán kívüli él, amely megsérti az optimalitási feltételt **do**
        a *belépő* él kiválasztása
        a belépő él hozzávétele a feszítőfához, a kialakult kör mentén javítás,
          és a *kilépő* él meghatározása
        a feszítőfa-szerkezet, a folyamértékek és a potenciálok frissítése
    end while
end procedure

11. ábra. Hálózati szimplex módszer

Az algoritmus implementációja a feszítőfa-szerkezet hatékony megvalósításán alapul, ugyanis ezzel biztosítjuk, hogy egy pivotálást gyorsan el lehessen végezni. Az ehhez kidolgozott különböző faindexeket és azok frissítésének módját részletesen bemutatja

Kennington és Helgason [46] könyve. Az egyik lehetséges módszer – amelyet mi is megvalósítottunk –, hogy minden csúcshoz nyilvántartjuk a rögzített gyökér csúcstól való távolságát (mélységét), egy mutatót a szülőcsúcsára, valamint egy olyan indexet, amely a fa egy adott mélységi bejárásában a következő csúcsot jelöli. Ezek felhasználásával $O(n)$ időben meghatározhatjuk a kilépő élt, és elvégezhetjük a szükséges frissítéseket.

A hálózati szimplex algoritmus tulajdonképpen a negatív kör módszer egy speciális változata, amelyben a nyilvántartott feszítőfa és a csúcspotenciálok segítségével $O(m)$ időben találhatunk negatív kört. Bizonyos esetekben viszont ezen nem tudunk pozitív folyammennyiséget folyatni (vagyis a kör valamely éle valójában nem része a maradék hálózatnak), így ezekben az ún. *elfajult* lépésekben nem tudunk javítani.

Nagy méretű minimális költségű folyam feladatokon végzett tesztek szerint egyes problémaosztályok esetén a pivotálások több mint 90 %-a elfajult lehet, és az algoritmus nem is lesz feltétlenül véges. Ezen problémák megoldására az ún. *szigorú megengedett feszítőfák* alkalmazása bizonyult a leghatékonyabbnak, amelyet Cunningham [16], valamint tőle függetlenül Barr, Glover és Klingman [7] javasolt. Egy megengedett feszítőfa-szerkezetet akkor nevezünk szigorúan megengedettnek, ha minden csúcsból küldhetünk pozitív folyammennyiséget a fa élein a gyökérbe a kapacitáskorlátok megsértése nélkül. A kilépő él alkalmas megválasztásával minden lépésben szigorú megengedett feszítőfát kapunk. Belátható, hogy ez a technika biztosítja az algoritmus végességét, valamint a gyakorlatban jelentősen csökkenti az elfajult pivotálások számát, de legrosszabb esetben ez továbbra is exponenciális lehet. Adhatók azonban olyan pivotálási szabályok, amelyekkel ez is elkerülhető. Cunningham [17], valamint később Goldfarb, Hao és Kai [37] is kidolgozott ilyen módszereket, amelyek többnyire azon alapulnak, hogy minden él csak bizonyos időközönként, periodikusan léphet be a bázisba.

3.4.1. Pivotálási szabályok

A hálózati szimplex algoritmus megvalósításának legkritikusabb része, hogy a pivotálások során milyen módszerrel választjuk ki a belépő élt. Egyrészt ez az egyes iterációk leglassabb, $O(m)$ idejű művelete, másrészt alapvetően befolyásolja az iterációk számát. Összesen öt különböző pivotálási szabályt valósítottunk meg, amelyeket az alábbiakban ismertetünk, a 4.2.6. szakaszban pedig bemutatjuk az ezeket összehasonlító mérési tesztjeink eredményét.

Legelső él választása

A legegyszerűbb stratégia az, hogy egy rögzített bejárási sorrend szerint mindig a legelső élt választjuk, amelyre nem teljesül az optimalitási feltétel. A gyakorlatban ezt úgy érdemes megvalósítani, hogy minden keresést az előző lépésben talált él után kezdünk, és a bejárás végére érve az első éltől folytatjuk. Ennek a módszernek a legfőbb előnye, hogy minden lépésben gyorsan talál belépő élt, viszont ezzel gyakran csak viszonylag kis mértékben tudunk javítani a célfüggvény értékén.

Legjobb él választása

Dantzig egy másik természetes módszert javasolt: minden lépésben válasszuk azt az élt, amelynek redukált költsége a legnagyobb mértékben sérti meg az optimalitási feltételt. Egy ilyen élen az egységnyi folyamhoz tartozó javítás maximális, ezért várhatóan ez a választás meglehetősen kevés iterációt eredményez, amit az általunk végzett tesztek eredményei is alátámasztottak. Ehhez viszont minden lépésben meg kell vizsgálnunk az összes élt, így a szabály összességében nem túl hatékony.

Blokkos keresés

A blokkos keresési szabály az előző két módszer egyfajta ötvözése, amelyet Grigoriadis [38] dolgozott ki. A cél az, hogy meglehetősen gyorsan, de egyúttal viszonylag jó belépő élt találjunk, ezért minden lépésben az élek rögzített méretű blokkjait nézzük át, és azokból választjuk ki a legjobb élt. Minden keresést az előző iterációban kiválasztott él után indítunk, és először csak egy blokknak megfelelő számú élt vizsgálunk meg, majd ha ezek között nincsen alkalmas él, akkor újabb blokkokkal folytatjuk. Ezáltal minden élt csak periodikusan választhatunk ki, ami Cunningham korábban említett vizsgálatai alapján általában jelentősen csökkenti az elfajult pivotálások számát.

A módszer egy fontos paramétere a blokkméret, amely egy kompromisszumot határoz meg a keresési idő és a megtalált éleken átlagosan elérhető javítás mértéke között. A 4.2.6. szakaszban bemutatott mérési eredményeink alapján $2\sqrt{m}$ bizonyult a legjobb választásnak (ahol m az élek száma).

Jelöltlista

Egy másik gyakran alkalmazott megközelítési mód az, hogy a választható (*jelölt*) élekből bizonyos időközönként listát építünk, és a következő pivotálások során ebből választunk. Mulvey [52] adta az első ilyen módszert, amelynek működését alapvetően két paraméter szabályozza: egyrészt megadjuk a jelöltlisták maximális méretét, másrészt azt, hogy legfeljebb hány iteráción keresztül használjuk őket. A felépített listákból mindig a legjobb élt választjuk ki, és közben töröljük azokat, amelyekre az előző pivotálások következtében már teljesül az optimalitási feltétel. Amikor a lista kiürül, vagy elérjük a megadott korlátot, akkor új listát építünk.

A tesztelés során úgy kaptuk a legjobb eredményeket, ha a listaméretet az élek számának 2 %-ára választottuk, és egy listát legfeljebb negyed ennyi iteráción keresztül használtunk.

Rendezett jelöltlista

Az előző módszer a gyakorlatban nem bizonyult hatékonynak, ezért kidolgoztuk egy javított változatát. Ebben a listákra csak olyan éleket veszünk fel, amelyek legalább egy adott mértékben megsértik az optimalitási feltételt, másrészt a felépített listákat rendezzük, így a következő iterációk során gyorsabban tudunk választani.

A mérési tesztjeink alapján a listák maximális hosszát az élek számának 1 %-ára választottuk, és a listaépítés során csak olyan éleket vettünk figyelembe, amelyek redukált költségének abszolút értéke legalább az eddig talált optimális érték 25 %-a.

4. Tesztelés

A megvalósított algoritmusok paramétereinek beállításához, az implementációs részletek kidolgozásához, valamint a különböző módszerek összehasonlításához számos sebességmérési tesztet végeztünk. A programokat openSUSE 10.2 operációs rendszer alatt, gcc 4.1.2 fordítóval, –O2 optimalizáció használatával fordítottuk le, és a *limetta.cs.elte.hu* szerveren futtattuk, amely két Intel© Xeon™ 3.20 GHz (2 MB cache) processzort és 2 GB memóriát tartalmaz.

4.1. Tesztadatok

A használt tesztfájlokat a NETGEN programmal hoztuk létre, amelyet 1974-ben Klingman, Napier és Stutz fejlesztett ki [49], és később számos algoritmus teszteléséhez alkalmazták. A program megadott paraméterek függvényében véletlen hálózatokat generál: először olyan éleket rögzít, amelyek egy megengedett folyamot biztosítanak, majd az így kapott gráfot bővíti véletlenszerűen.

A 2. táblázatban megadjuk a létrehozott tesztfájlcsoportokban szereplő hálózatok számát és méretét. Az STD megnevezésű csoportokban általános minimális költségű folyam feladatok szerepelnek, köztük az a leggyakrabban vizsgált 40 probléma, amelyet a NETGEN program készítő tettek közzé. Ezen csoportokban a termelő és fogyasztó csúcsok száma általában 100 és 1000 között változik, az össztermelés többnyire 1–4 millió, az élek kapacitása pedig ezekhez igazodik. Az SPR és DEN hálózatokat az STD3 és STD4 osztályból válogattuk ki: az SPR csoport 6 ritka gráfból áll, amelyekre $2n \leq m \leq 5n$, a DEN csoport pedig 4 sűrűbb gráfból, amelyekre $30n \leq m \leq 50n$. A TRA csoport 14 szállítási feladatot tartalmaz (l. 1.2.3. szakasz), az ASG csoport pedig 8 hozzárendelési feladatot (l. 1.2.4. szakasz), tehát mindkettőben páros gráfok szerepelnek, továbbá az utóbbi esetben minden termelés/fogyasztás érték és kapacitás egy.

Csoport	*db*	*Csúcsok száma (n)*	*Élek száma (m)*
STD1	20	200 – 400	1000 – 5000
STD2	15	750 – 1500	3000 – 10 000
STD3	15	3000 – 8000	15 000 – 80 000
STD4	12	10 000 – 20 000	50 000 – 500 000
SPR	6	5000 – 15 000	15 000 – 50 000
DEN	4	10 000 – 15 000	300 000 – 500 000
TRA	14	400 – 1000	10 000 – 50 000
ASG	8	1000 – 2000	5000 – 25 000

2. táblázat. Tesztadatok

4.2. Algoritmusok változatai

A következőkben bemutatjuk és összehasonlítjuk a megvalósított algoritmusok különböző paraméterértékeinek és módosításainak gyakorlatban mért hatását. A megadott táblázatokban mindig az egyes csoportok fájljaira kapott *átlagos futási idők* szerepelnek *másodpercben* mérve (a fájl beolvasását és a hálózat felépítését nem számítva).

4.2.1. Megengedett folyam keresése

A 3. táblázatban összehasonlítjuk a megengedett folyam keresésére adott két megoldási módszert, amelyeket a 3.2.1. szakaszban tárgyaltunk. A megvalósításhoz a LEMON könyvtár *Preflow* és *Circulation* osztályait használtuk, amelyek nem ezen fejlesztés keretében készültek, viszont a vizsgált problémára a negatív kör algoritmusokban alkalmaztuk őket.

	Gráfmásolás és -átalakítás	*Preflow*	*Másolás és Preflow*	*Circulation*
STD1	0,0007	0,0003	0,0010	**0,0004**
STD2	0,0015	0,0009	0,0024	**0,0011**
STD3	0,0102	0,0130	0,0232	**0,0106**
STD4	0,0831	0,0584	0,1415	**0,1153**
TRA	0,0073	0,0031	0,0104	**0,0045**
ASG	0,0043	0,0008	0,0051	**0,0017**
Összesen:	0,1071	0,0765	0,1836	**0,1336**

3. táblázat. Megengedett folyam keresése

4.2.2. Egyszerű negatív kör algoritmus

A 4. táblázatban bemutatjuk a 3.2.2. szakaszban ismertetett negatív kör algoritmus megvalósítása során alkalmazott javítások hatását.

A) A Bellman–Ford-algoritmus iterációinak számát nem korlátozzuk (mindig n iterációt végzünk), és minden lépésben csak egy kör mentén javítunk.

B) A Bellman–Ford-algoritmus iterációinak számát korlátozzuk ($S = 2$, $\alpha = \frac{3}{2}$), és minden lépésben csak egy kör mentén javítunk.

C) A Bellman–Ford-algoritmus iterációinak számát korlátozzuk ($S = 2$, $\alpha = \frac{3}{2}$), és minden lépésben az összes megtalált kör mentén javítunk.

	A	B	C
STD1	111,76	0,28	**0,28**
STD2	6261,51	3,58	**3,44**
ASG	52,12	0,39	**0,29**
Összesen:	6425,39	4,25	**4,01**

4. táblázat. Az egyszerű negatív kör algoritmus javításainak hatása

S és α megválasztására számos kombinációt kipróbáltuk, az 5. és 6. táblázatban néhány jellemző értéket kiválasztva külön-külön szemléltetjük a két paraméter hatását. Egyértelműen $S = 2$, $\alpha = \frac{3}{2}$ bizonyult a legjobbnak, ezért használtuk a 4. táblázatban ezt a változatot, és a továbbiakban bemutatott összehasonlításokban is ez szerepel.

	$\alpha = \frac{5}{4}$	$\alpha = \frac{3}{2}$	$\alpha = \frac{7}{4}$	$\alpha = 2$	$\alpha = \frac{5}{2}$	$\alpha = 3$
STD1	0,29	**0,28**	0,29	0,31	0,36	0,41
STD2	3,86	**3,44**	4,73	4,49	6,98	10,64
ASG	0,33	0,29	**0,28**	0,31	0,31	0,34
Összesen:	4,48	**4,01**	5,30	5,11	7,65	11,39

5. táblázat. Az α paraméter hatása $S = 2$ esetén

	$S = 1$	$S = 2$	$S = 3$	$S = 4$
STD1	**0,25**	0,28	0,35	0,53
STD2	3,50	**3,44**	5,23	11,17
ASG	0,31	**0,29**	0,29	0,31
Összesen:	4,06	**4,01**	5,87	12,01

6. táblázat. Az S paraméter hatása $\alpha = \frac{3}{2}$ esetén

4.2.3. Minimális átlagú negatív kör algoritmus

A 3.2.3. szakaszban bemutatott minimális átlagú negatív kör algoritmus a gyakorlatban kifejezetten lassúnak bizonyult, annak ellenére, hogy a minimális átlagú kör keresésére Karp algoritmusát hatékonyan megvalósítottuk. A 7. táblázatban az előző szakaszban

vizsgált egyszerű negatív kör algoritmussal hasonlítjuk össze: a javítások nélküli változatnál ugyan jóval gyorsabb (mivel kevesebb iterációt végez), a javított változatnál azonban sokkal lassabb.

	Minimális átlagú negatív kör alg.	*Egyszerű negatív kör algoritmus*	
		Javítások nélkül	*Javításokkal*
STD1	19,82	111,76	**0,28**
STD2	628,76	6261,51	**3,44**
ASG	119,81	52,12	**0,29**
Összesen:	768,39	6425,39	**4,01**

7. táblázat. A negatív kör algoritmusok összehasonlítása

4.2.4. Ismételt legrövidebb út algoritmus

A 8. táblázat a 3.3.1. szakaszban ismertetett ismételt legrövidebb út algoritmus legfontosabb javításának hatását mutatja be: az egyes lépésekben a Dijkstra-algoritmust csak az első hiánnyal rendelkező csúcs eléréséig futtatjuk.

	Teljes futtatás	*Első találatig*	*Arány*
STD1	0,065	**0,015**	23,08 %
STD2	0,163	**0,047**	28,83 %
STD3	33,274	**3,248**	9,76 %
TRA	5,131	**0,548**	10,68 %
ASG	0,658	**0,026**	3,95 %
Összesen:	39,291	**3,884**	9,89 %

8. táblázat. Dijkstra-algoritmus futtatása az első találatig

4.2.5. Kapacitásskálázó algoritmus

A 3.3.2. szakaszban tárgyalt kapacitásskálázó algoritmus eredeti és általunk kidolgozott változatát a 9. táblázatban hasonlítjuk össze.

A Dijkstra-algoritmust ebben a módszerben is csak az első megfelelő csúcs eléréséig futtatjuk, aminek hatását a 10. táblázatban szemléltetjük. Itt is ez bizonyult a legjelentősebb javításnak.

A 11. táblázat pedig a K skálázási paraméter szerepét mutatja: átlagosan $K = 2$ és $K = 3$ esetén kaptuk a legjobb futási időket. A továbbiakban mindig a $K = 2$ értéket használjuk.

	Eredeti algoritmus	*Módosított változat*	*Arány*
STD1	0,018	**0,015**	83,33 %
STD2	0,049	**0,038**	77,55 %
STD3	1,545	**1,075**	69,58 %
STD4	15,206	**12,381**	81,42 %
TRA	0,344	**0,302**	87,79 %
ASG	0,031	**0,027**	87,10 %
Összesen:	17,193	**13,838**	80,49 %

9. táblázat. A kapacitásskálázó algoritmus módosításának hatása

	Teljes futtatás	*Első találatig*	*Arány*
STD1	0,078	**0,015**	19,23 %
STD2	0,141	**0,038**	27,66 %
STD3	18,486	**1,075**	5,88 %
TRA	2,227	**0,302**	13,56 %
ASG	0,484	**0,027**	5,58 %
Összesen:	21,416	**1,457**	6,80 %

10. táblázat. Dijkstra-algoritmus futtatása az első találatig

	$K = 2$	$K = 3$	$K = 4$	$K = 5$	$K = 6$	$K = 8$
STD1	0,015	0,013	0,013	**0,011**	0,014	0,015
STD2	0,038	0,034	**0,033**	0,034	0,034	0,035
STD3	**1,075**	1,085	1,160	1,123	1,201	1,292
TRA	0,302	0,298	0,321	0,304	**0,286**	0,301
ASG	**0,027**	0,028	0,027	0,027	0,028	0,027
Összesen:	**1,457**	1,458	1,554	1,499	1,563	1,670

11. táblázat. A skálázási paraméter hatása

4.2.6. Hálózati szimplex algoritmus

A 12. táblázatban, valamint a 12. és 13. ábrán összehasonlítjuk a 3.4. alfejezetben bemutatott hálózati szimplex algoritmus különböző pivotálási szabályokon alapuló változatait. Minden esetben a leghatékonyabbnak talált implementációt és a legjobb paraméterértékeket alkalmaztuk.

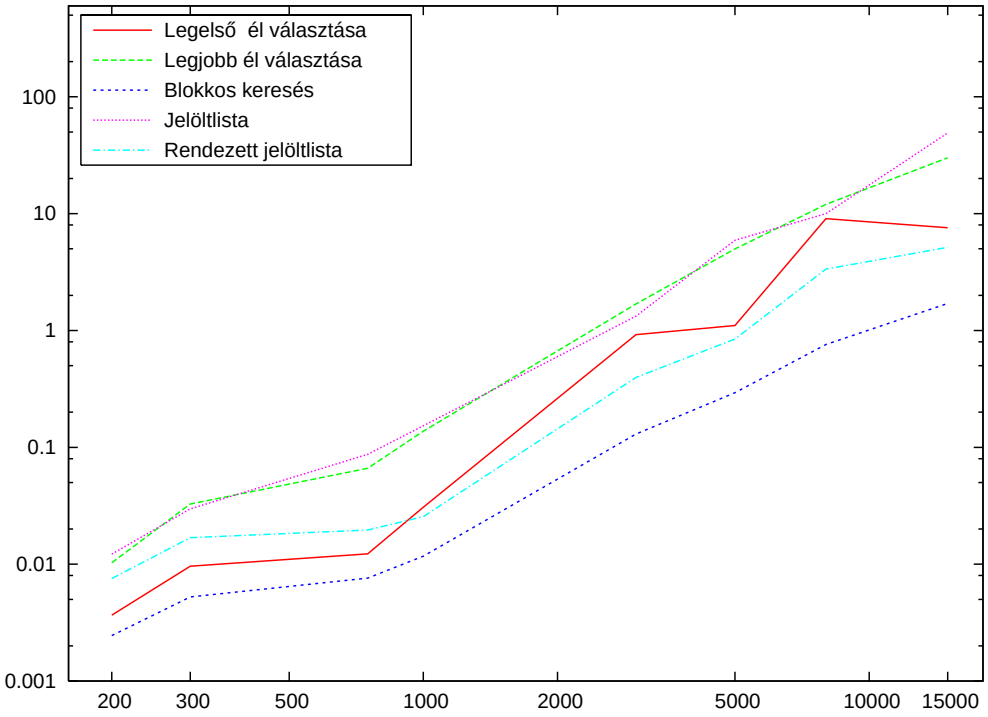
	Legelső él választása	*Legjobb él választása*	*Blokkos keresés*	*Jelöltlista*	*Rendezett jelöltlista*
STD1	0,006	0,010	**0,004**	0,022	0,008
STD2	0,024	0,069	**0,017**	0,200	0,033
STD3	0,935	2,278	**0,355**	5,766	0,880
STD4	3,229	53,083	**1,919**	111,211	2,988
SPR	1,948	4,493	**0,799**	13,178	2,011
DEN	3,139	58,801	**1,664**	120,795	2,564
TRA	0,085	0,252	**0,045**	0,404	0,089
ASG	0,041	0,218	**0,029**	0,594	0,046
Összesen:	9,407	119,204	**4,832**	252,170	8,619

12. táblázat. Pivotálási szabályok összehasonlítása

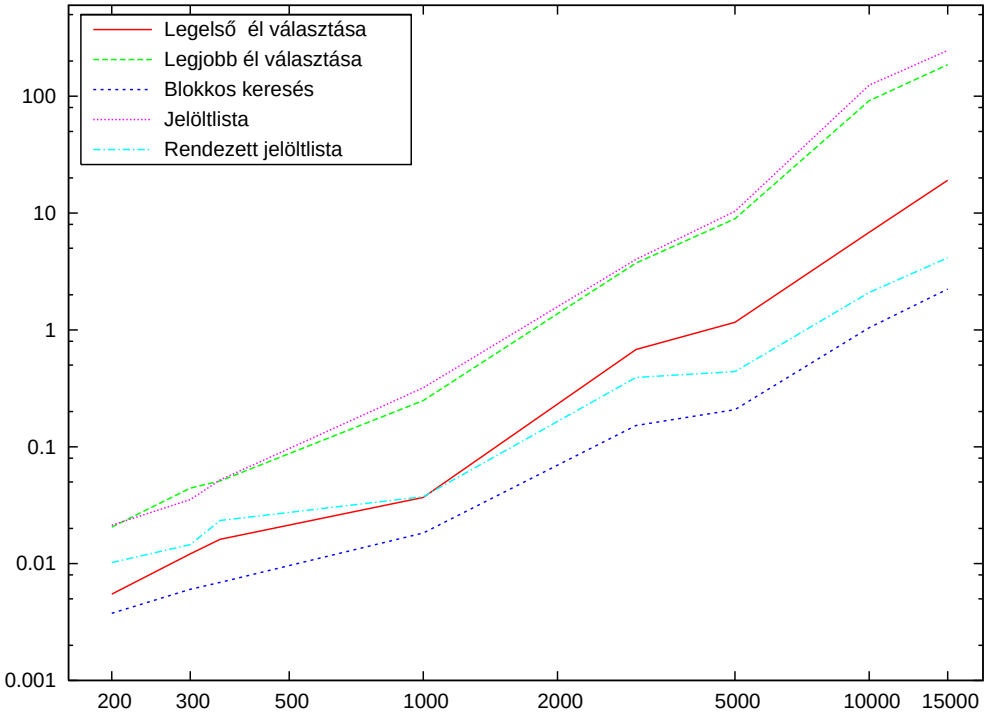
Egyszerűsége ellenére egyértelműen a blokkos keresés bizonyult a leghatékonyabb pivotálási szabálynak, ezért számos tesztet végeztünk a blokkméret beállításához. Grigoriadis [38] az élek számának 1–8,5 %-át javasolta, mi azonban a különböző méretű hálózatokra nagyon eltérő arány esetén kaptuk a legjobb futási időket, ami a 13. táblázatban is látható. Azt találtuk, hogy az optimális blokkméret nem az élek számával, hanem inkább annak négyzetgyökével arányos, a legjobb eredményeket pedig $\lfloor 2\sqrt{m} \rfloor$ esetén kaptuk.

	$m/50$	$m/100$	$m/200$	$m/300$	$\lfloor 2\sqrt{m} \rfloor$
STD1	**0,004**	0,004	0,005	0,005	**0,004**
STD2	**0,017**	0,018	0,021	0,025	**0,017**
STD3	0,357	**0,338**	0,408	0,490	0,355
STD4	2,954	2,260	1,922	**1,885**	1,919
SPR	0,954	**0,787**	0,865	1,030	0,799
DEN	2,756	2,014	**1,652**	1,666	1,664
TRA	**0,045**	0,046	0,052	0,055	**0,045**
ASG	**0,028**	0,031	0,036	0,041	0,029
Összesen:	7,115	5,498	4,961	5,197	**4,832**

13. táblázat. Blokkméret hatása blokkos keresés esetén



12. ábra. Pivotálási szabályok összehasonlítása (logaritmikus skála, $3n \leq m \leq 8n$)



13. ábra. Pivotálási szabályok összehasonlítása (logaritmikus skála, $8n \leq m \leq 25n$)

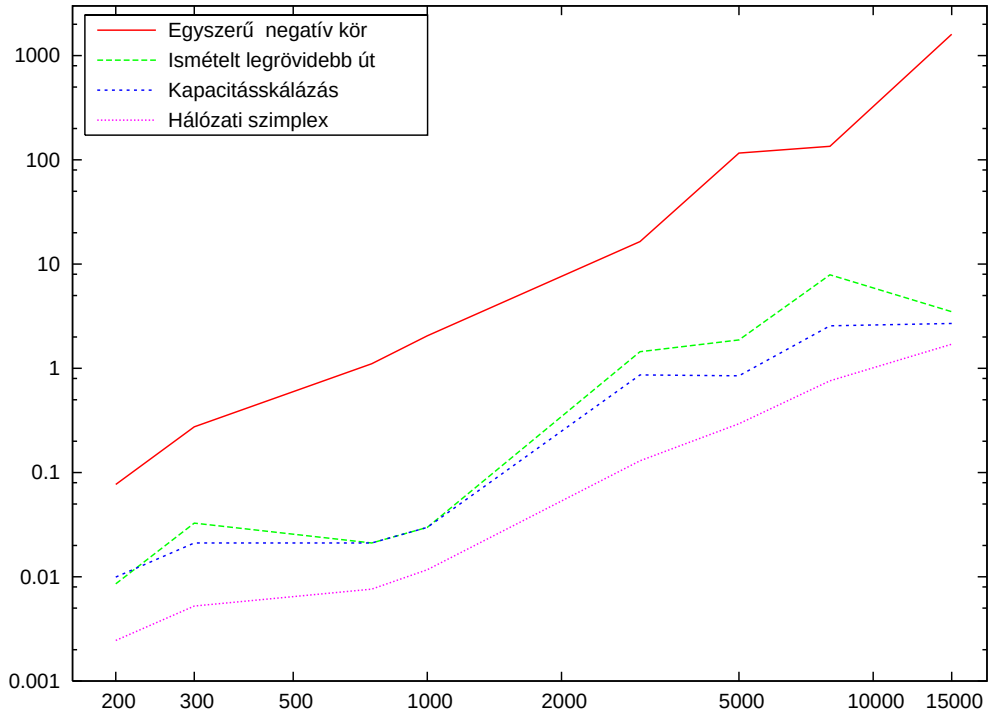
4.3. Összehasonlítás

A 14. táblázatban a megvalósított öt algoritmus legjobb változatát hasonlítjuk össze. Egyértelműen a hálózati szimplex módszer bizonyult a leghatékonyabbnak, de a duál algoritmusok is meglehetősen gyorsak, sőt a hozzárendelési feladatokra (ASG), ahol az összetermelés, illetve az élek kapacitása alacsony, valamivel gyorsabbak a hálózati szimplexnél. A negatív kör módszerek viszont sokkal lassabbak, ezért a nagyobb hálózatokra nem is futtattuk le őket.

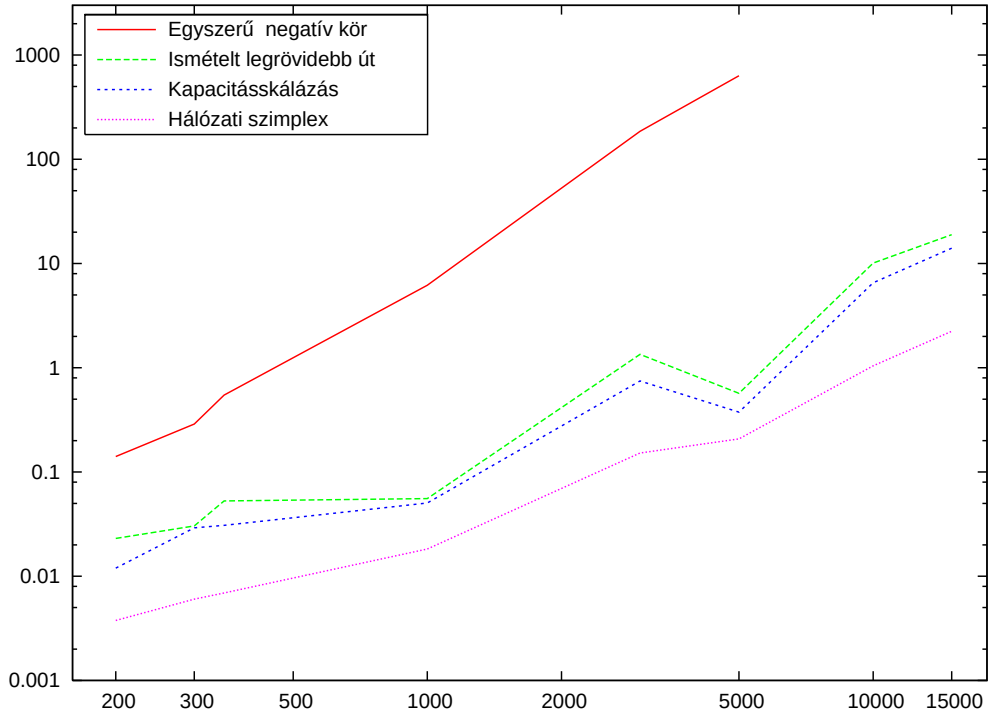
	Egyszerű negatív kör	*Min. átlagú negatív kör*	*Ismételt legrövidebb út*	*Kapacitás-skálázás*	*Hálózati szimplex*
STD1	0,28	19,82	0,015	0,015	**0,004**
STD2	3,44	628,76	0,047	0,038	**0,017**
STD3	188,44	–	3,248	1,075	**0,355**
STD4	–	–	15,869	12,381	**1,919**
SPR	–	–	4,261	1,858	**0,799**
DEN	–	–	19,887	9,211	**1,664**
TRA	–	–	0,548	0,302	**0,045**
ASG	0,29	119,81	**0,026**	0,027	0,029
Összesen:			43,901	24,907	**4,832**

14. táblázat. Algoritmusok összehasonlítása

A 14. és 15. ábrán szintén az egyes módszerek futási idejét hasonlítjuk össze – a minimális átlagú negatív kör algoritmus kivételével – különböző sűrűségű hálózatokon. Mindkét grafikon a csúcsok számának függvényében ábrázolja a másodpercben mért futási időket. (A használt tesztfájlokat az STD csoportokból válogattuk ki.)



14. ábra. Algoritmusok összehasonlítása (logaritmikus skála, $3n \leq m \leq 8n$)



15. ábra. Algoritmusok összehasonlítása (logaritmikus skála, $8n \leq m \leq 25n$)

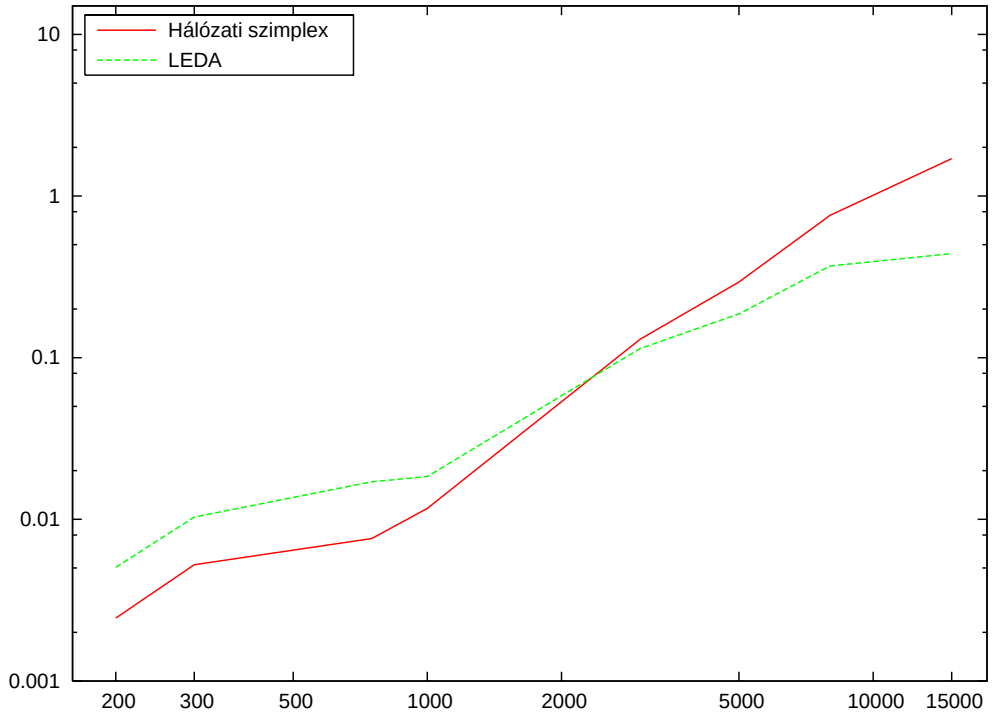
A fejlesztés során különösen fontos szempont volt az is, hogy a megvalósított módszereket összehasonlítsuk mások által adott implementációkkal. Erre a célra egyértelműen a széles körben ismert és alkalmazott LEDA programcsomag (*www.algorithmic-solutions.com*) volt a legalkalmasabb, amely a LEMON-hoz hasonlóan számos adatszerkezet és algoritmus hatékony megvalósítását tartalmazza, használata azonban nem ingyenes. Az összehasonlítások során a LEDA 5.0 verziót használtuk, és az ezzel készült programot ugyanolyan feltételek mellett fordítottuk és teszteltük, mint a saját algoritmusokat.

A 15. táblázatban megadjuk a LEDA könyvtárban megvalósított minimális költségű folyam eljárás futási idejét az egyes problémaosztályokra, összevetve az általunk adott hálózati szimplex implementációval. Ezen eredmények alapján megállapíthatjuk, hogy a hálózati szimplex a legtöbb esetben hatékonyabb. Az összehasonlításban a legmeghatározóbb paraméternek a gráf sűrűsége bizonyult: a nagyméretű és kifejezetten ritka hálózatokon, ahol az élek száma legfeljebb a csúcsok számának 5–10-szerese (STD3, SPR), a LEDA hatékonyabb, a kisebb méretű (STD1, STD2, TRA, ASG), valamint a sűrűbb (STD4, DEN) gráfokon viszont a hálózati szimplex algoritmus jóval gyorsabb.

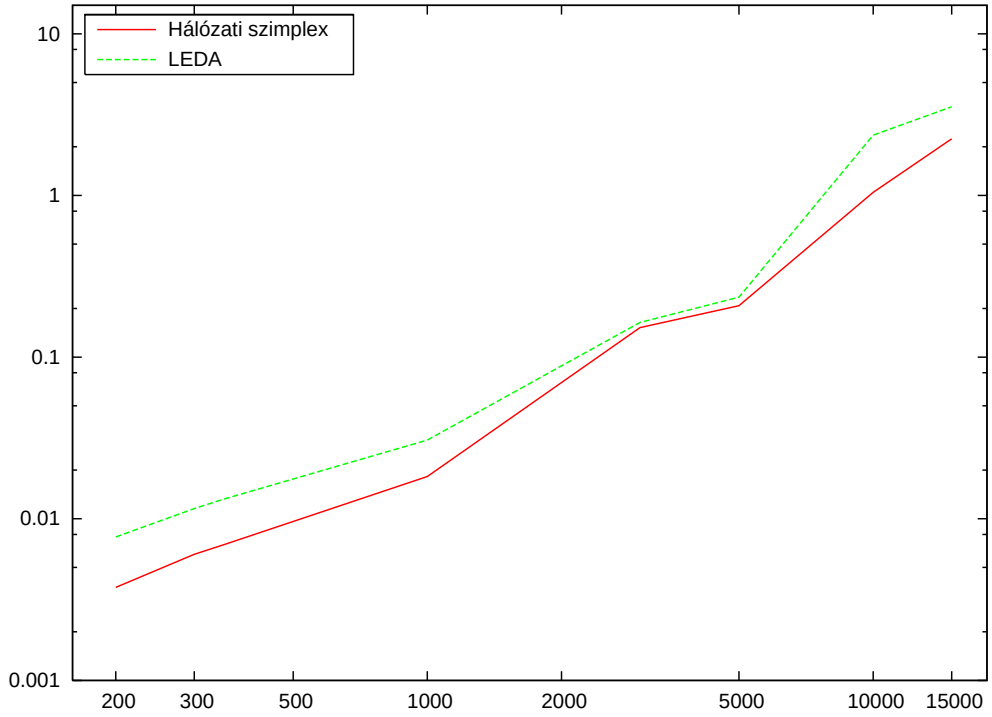
	Hálózati szimplex	*LEDA*
STD1	**0,004**	0,007
STD2	**0,017**	0,024
STD3	0,355	**0,230**
STD4	**1,919**	2,578
SPR	0,799	**0,309**
DEN	**1,664**	3,424
TRA	**0,045**	0,084
ASG	**0,029**	0,037
Összesen:	**4,832**	6,693

15. táblázat. Hálózati szimplex – LEDA összehasonlítás

A 16. és 17. ábrán adott grafikonokkal a hálózati szimplex algoritmusnak és a LEDA eljárásának futási idejét szemléltetjük különböző sűrűségű gráfokon.



16. ábra. Hálózati szimplex – LEDA összehasonlítás (log. skála, $3n \leq m \leq 8n$)



17. ábra. Hálózati szimplex – LEDA összehasonlítás (log. skála, $8n \leq m \leq 25n$)

5. Összefoglalás

Az alábbiakban összefoglaljuk munkánk legfontosabb eredményeit.

- Mélyrehatóan vizsgáltuk a minimális költségű folyam modell elméleti hátterét és a különböző megoldási módszereket.
- Három alapvetően különböző megközelítési módot alkalmazva hatékonyan megvalósítottunk öt algoritmust, illetve azok több változatát.
- Kidolgoztunk egy egyszerű negatív kör algoritmust, és a tesztelési eredmények alapján meghatároztuk a paramétereinek optimális értékét.
- Megvalósítottuk a minimális átlagú negatív kör algoritmust, amely erősen polinomiális időben fut, a gyakorlatban viszont sokkal lassabbnak találtuk a többi módszernél.
- Az ismételt legrövidebb út módszert és a kapacitásskálázó algoritmust jelentős javításokkal valósítottuk meg. Egyszerűségük ellenére mindkét implementáció meglehetősen hatékonynak bizonyult.
- A hálózat szimplex algoritmushoz megvalósítottunk és összehasonlítottunk többféle pivotálási stratégiát. A legjobb futási időket a blokkos keresés, valamint a rendezett jelöltlistákat használó, általunk kidolgozott szabály eredményezte. Az előbbi módszer alapos tesztelése alapján arra a következtetésre jutottunk, hogy az általános gyakorlattal ellentétben a blokkméretet nem az élek számával, hanem annak négyzetgyökével arányosan érdemes megválasztani.
- A fenti szabályt alkalmazva egyértelműen a hálózati szimplex algoritmus bizonyult a leghatékonyabbnak az általunk megvalósított módszerek közül. A legtöbb esetben gyorsabb a széles körben alkalmazott LEDA programkönyvtár minimális költségű folyam eljárásánál is. Általában elmondható, hogy a nagyméretű és elég ritka gráfokon hatékonyabb a LEDA, ellenben a kisebb hálózatok, valamint a nagyméretű, de sűrűbb hálózatok esetén a hálózati szimplex gyorsabb.

A fejlesztés során készült C++ forrásfájlok bekerültek a LEMON programkönyvtárba (`http://lemon.cs.elte.hu`).

Irodalomjegyzék

[1] Ravindra K. Ahuja – Andrew V. Goldberg – James B. Orlin – Robert E. Tarjan: Finding minimum-cost flows by double-scaling. Technical report, Stanford University, Stanford, CA, 1988.

[2] Ravindra K. Ahuja – Thomas L. Magnanti – James B. Orlin: *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., February 1993.

[3] Ravindra K. Ahuja – Kurt Mehlhorn – James B. Orlin – Robert E. Tarjan: Faster algorithms for the shortest path problem. *Journal of the ACM*, 37(2):213–223, 1990.

[4] Ravindra K. Ahuja – James B. Orlin – Robert E. Tarjan: Improved time bounds for the maximum flow problem. *SIAM Journal on Computing*, 18(5):939–954, 1989.

[5] Ronald D. Armstrong – Darwin Klingman – David Whitman: Implementation and analysis of a variant of the dual method for the capacitated transshipment problem. *The European Operations Research Journal*, 4:403–420, 1980.

[6] Francisco Barahona – Éva Tardos: Note on Weintraub's minimum cost circulation algorithm. *SIAM Journal on Computing*, 18:579–583, 1989.

[7] Richard S. Barr – Fred Glover – Darwin Klingman: The alternating basis algorithm for assignment problems. *Mathematical Programming*, 13:1–13, 1977.

[8] Dimitri P. Bertsekas: A distributed algorithm for the assignment problem. Working Paper, Laboratory for Information and Decision Systems, MIT, 1979.

[9] Dimitri P. Bertsekas: A unified framework for primal–dual methods in minimum cost network flow problems. *Mathematical Programming*, 32:125–145, 1985.

[10] Dimitri P. Bertsekas – Paul Tseng: Relaxation methods for minimum cost ordinary and generalized network flow problems. *Operations Research*, 36(1):93–114, 1988.

[11] Robert G. Bland – David L. Jensen: On the computational behavior of a polynomial-time network flow algorithm. Technical Report 661, School of Operations Research and Industrial Engineering, Cornell University, 1985. *Mathematical Programming*, 54(1):1–39, 1992.

[12] Gordon Bradley – Gerald Brown – Glenn Graves: Design and implementation of large scale primal transshipment algorithms. *Management Science*, 24(1):1–34, 1977.

[13] Robert G. Busacker – Paul J. Gowen: A procedure for determining a family of minimum-cost network flow patterns. Technical Report 15, Operations Research Office, The Johns Hopkins University, Bethesda, Maryland, 1960.

[14] Boris V. Cherkassky – Andrew V. Goldberg: Negative-cycle detection algorithms. In *European Symposium on Algorithms*, pages 349–363, 1996.

[15] Thomas H. Cormen – Charles E. Leiserson – Ronald L. Rivest – Clifford Stein: *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

[16] William H. Cunningham: A network simplex method. *Mathematical Programming*, 11:105–116, 1976.

[17] William H. Cunningham: Theoretical properties of the network simplex method. *Mathematics of Operations Research*, 4:196–208, 1979.

[18] Ali Dasdan – Rajesh K. Gupta: Faster maximum and minimum mean cycle alogrithms for system performance analysis. Technical Report ICS-TR-97-07, IEEE Transactions on Computer-Aided Design, 1997.

[19] Ali Dasdan – Sandra S. Irani – Rajesh K. Gupta: An experimental study of minimum mean cycle algorithms. Technical Report UCI-ICS #98-32, University of California, Irvine, CA, 1998.

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

[21] Jack Edmonds – Richard M. Karp: Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.

[22] Lester R. Ford – Delbert R. Fulkerson: *Flows in Networks*. Princeton University Press, 1962.

[23] Michael L. Fredman – Robert E. Tarjan: Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.

[24] Satoru Fujishige: A capacity-rounding algorithm for the minimum-cost circulation problem: A dual framework of the Tardos algorithm. *Mathematical Programming*, 35(3):298–308, 1986.

[25] Delbert R. Fulkerson: An out-of-kilter method for minimal-cost flow problems. *Journal of the Society for Industrial and Applied Mathematics*, 9:18–27, 1961.

[26] Zvi Galil – Éva Tardos: An $O(n^2 \log n(m + n \log n))$ min-cost flow algorithm. *Journal of the ACM*, 35(2):374–386, 1988.

[27] Michael R. Garey – David S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., January 1979.

[28] Fred Glover – David Karney – Darwin Klingman – Albert Napier: A computational study on start procedures, basis change criteria, and solution algorithms for transportation problems. *Management Science*, 20(5):793–813, 1974.

[29] Andrew V. Goldberg: An efficient implementation of a scaling minimum-cost flow algorithm. Technical report, Departement of Computer Science, Stanford University, 1992. *Journal of Algorithms*, 22(1):1–29, 1997.

[30] Andrew V. Goldberg – Satish Rao: Beyond the flow decomposition barrier. *Journal of the ACM*, 45(5):783–797, 1998.

[31] Andrew V. Goldberg – Robert E. Tarjan: A new approach to the maximum flow problem. In *STOC '86: Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 136–146, New York, NY, 1986. ACM Press.

[32] Andrew V. Goldberg – Robert E. Tarjan: Solving minimum-cost flow problems by successive approximation. In *STOC '87: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 7–18, New York, NY, 1987. ACM Press.

[33] Andrew V. Goldberg – Robert E. Tarjan: Finding minimum-cost circulations by canceling negative cycles. In *STOC '88: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 388–397, New York, NY, 1988. ACM Press.

[34] Andrew V. Goldberg – Robert E. Tarjan: Finding minimum-cost circulations by successive approximation. *Mathematics of Operations Research*, 15(3):430–466, 1990.

[35] Andrew V. Goldberg – Éva Tardos – Robert E. Tarjan: Network flow algorithms. In B. Korte, L. Lovász, H. J. Prömel, A. Schrijver, editors, *Paths, Flows and VLSI-Layout*, pages 101–164. Springer-Verlag, 1990.

[36] Donald Goldfarb – Jianxiu Hao: Polynomial simplex algorithms for the minimum cost network flow problem. *Algorithmica*, 8:145–160, 1992.

[37] Donald Goldfarb – Jianxiu Hao – Sheng-Roan Kai: Anti-stalling pivot rules for the network simplex algorithm. *Networks*, 20(1):79–91, 1990.

[38] Michael D. Grigoriadis: An efficient implementation of the network simplex method. *Mathematical Programming Study*, 26:83–111, 1986.

[39] Richard V. Helgason – Jeffery L. Kennington: An efficient procedure for implementing a dual simplex network flow algorithm. *AIEE Transactions*, 9(1):63–68, 1977.

[40] Frank L. Hitchcock: The distribution of a product from several sources to numerous localities. *Journal of Maths Physics*, 20:224–230, 1941.

[41] Masao Iri: A new method for solving transportation-network problems. *Journal of the Operations Research Society of Japan*, 3:27–87, 1960.

[42] William S. Jewell: Optimal flow through networks. Technical Report 8, Operations Research Center, MIT, Cambridge, MA, 1958.

[43] Leonid V. Kantorovich: Mathematical methods in the organization and planning of production, 1939. Translated in *Management Science*, 6:336–422, 1960.

[44] Richard M. Karp: A characterization of the minimum cycle mean in a digraph. *Discrete Math.*, 23:309–311, 1978.

[45] Damian J. Kelly – Garrett M. O'Neill: The minimum cost flow problem and the network simplex method. Master's thesis, University College, Dublin, 1991. `http://www.derekroconnor.net/home/PAPERS/MMS-91.pdf`

[46] Jeffery L. Kennington – Richard V. Helgason: *Algorithms for Network Programming*. John Wiley & Sons, Inc., New York, NY, 1980.

[47] Valerie King – Satish Rao – Robert E. Tarjan: A faster deterministic maximum flow algorithm. *Journal of Algorithms*, 17:447–474, 1994.

[48] Morton Klein: A primal method for minimal cost flows with applications to the assignment and transportation problems. *Management Science*, 14:205–220, 1967.

[49] Darwin Klingman – Albert Napier – Joel Stutz: NETGEN: A program for generating large scale capacitated assignment, transportation, and minimum cost flow network problems. *Management Science*, 20:814–821, 1974.

[50] Donald E. Knuth: *Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley Professional, 3rd edition, July 1997.

[51] Harold W. Kuhn: The hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2:83–97, 1955.

[52] John M. Mulvey: Pivot strategies for primal simplex network codes. *Journal of the ACM*, 25:266–270, 1978.

[53] James B. Orlin: Genuinely polynomial simplex and non-simplex algorithms for the minimum cost flow problem. Technical Report No. 1615-84, Sloan School of Management, MIT, Cambridge, MA, 1984.

[54] James B. Orlin: A faster strongly polynomial minimum cost flow algorithm. In *STOC '88: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 377–387, New York, NY, 1988. ACM Press.

[55] James B. Orlin: A polynomial time primal network simplex algorithm for minimum cost flows. *Mathematical Programming*, 78(1):109–129, 1997.

[56] James B. Orlin – Serge A. Plotkin – Éva Tardos: Polynomial dual network simplex algorithms. *Mathematical Programming*, 60(3):255–276, 1993.

[57] Serge A. Plotkin – Éva Tardos: Improved dual network simplex. In *SODA '90: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 367–376, Philadelphia, PA, 1990. Society for Industrial and Applied Mathematics.

[58] Hans Röck: Scaling techniques for minimal cost network flows. In U. Pape, editor, *Discrete Structures and Algorithms*, pages 181–191, München, 1980. Carl Hanser.

[59] Alexander Schrijver: Paths and flows a historical survey. *CWI Quarterly*, 6(3):169–183, 1993.

[60] Alexander Schrijver: *Combinatorial Optimization: Polyhedra and Efficiency*. Springer-Verlag, 2003.

[61] Daniel D. Sleator – Robert E. Tarjan: A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

[62] Venkatesh Srinivasan – Gerald L. Thompson: Benefit-cost analysis of coding techniques for the primal transportation algorithm. *Journal of the ACM*, 20(2):194–213, 1973.

[63] Robert E. Tarjan: *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, 1983.

[64] Robert E. Tarjan: Efficiency of the primal network simplex algorithm for the minimum-cost circulation problem. *Mathematics of Operations Research*, 16(2):272–291, 1991.

[65] Robert E. Tarjan: Dynamic trees as search trees via euler tours, applied to the network simplex algorithm. *Mathematical Programming*, 78:169–177, 1997.

[66] Mikkel Thorup: Integer priority queues with decrease key in constant time and the single source shortest paths problem. In *STOC '03: Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, pages 149–158, New York, NY, 2003. ACM Press.

[67] Éva Tardos: A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–255, 1985.

[68] Andrés Weintraub: A primal algorithm to solve network flow problems with convex costs. *Management Sciences*, 21(1):87–97, 1974.

[69] Norman Zadeh: A bad network problem for the simplex method and other minimum cost flow algorithms. *Mathematical Programming*, 5(1):255–266, 1973.