PRO PATRIA AD MORTEM

Döntéselőkészítési módszerek

Alapfeladatok és alkalmazások

SZÁSZI GÁBOR
TÓTH BENCE

Dialóg Campus

Döntéselőkészítési módszerek

Pro patria ad mortem

Szászi Gábor – Tóth Bence

DÖNTÉSELŐKÉSZÍTÉSI MÓDSZEREK

Alapfeladatok és alkalmazások

Dialóg Campus ❖ Budapest, 2019

Szakmai lektor
Dr. Fábos Róbert

Tartalom

Előszó

Több mint tíz év telt el azóta, hogy a Nemzeti Közszolgálati Egyetemen, pontosabban egyik elődintézményében, a Zrínyi Miklós Nemzetvédelmi Egyetemen egyetemi jegyzet készült az operációkutatás témakörében. De nemcsak a képzési struktúra ez idő alatt bekövetkezett változásai, hanem a mindennapi használat részévé vált újabb módszerek és alkalmazási területek bemutatásának szükségessége is indokolta egy korszerű jegyzet kiadását. Ennek a célnak kíván jelen jegyzet megfelelni, ugyanis célirányosan az NKE Hadtudományi és Honvédtisztképző Karon, a katonai logisztika BSc szakon oktatott *Döntéselőkészítési*[1] *módszerek* kurzus tananyagának lefedésére készült.

A jegyzet készítése során célul tűztük ki, hogy a hallgatók megismerjék az operációkutatás alapfogalmait, módszertanát, valamint a szállítás (polgári és katonai) szervezésében, irányításában hasznosan alkalmazható operációkutatási modellek felépítését és megoldási algoritmusát. Ebből következik, hogy a hangsúlyt nem a matematikai tételek ismertetésére, bizonyítására helyeztük, hanem az egyes modellek sajátosságainak, a jelentkező problémák, feladatok megoldási lehetőségeinek, a modellek gyakorlatban történő alkalmazásának bemutatására. A jegyzetben már nem tárgyalt egyéb modellekről az irodalomjegyzékben feltüntetett szakirodalmak részletes ismeretet nyújtanak.

A jegyzetben arra törekedtünk, hogy konkrét példákon mutassuk be az operációkutatási modellek katonai téren való alkalmazási lehetőségeit, kiemelten tárgyalva a szállítási feladatok optimalizálására szolgáló rendszereket. Feltételeztük a mátrixalgebra, a valószínűségszámítás és a matematikai statisztika alapvető ismeretét. A leglényegesebb angol szakkifejezéseket az első szövegközi előfordulásukkor külön megjelenítjük.

A szerzők ezúton fejezik ki köszönetüket Jurás Katalinnak és Nagy Kornélnak a jegyzet elkészítéséhez nyújtott pótolhatatlan segítségükért.

Budapest, 2018. június 30.

A szerzők

---

[1] A kiadó megjegyzése: a kifejezés az MTA helyesírási szabályainak megfelelő *döntés-előkészítés* írásmód helyett a szakmai hagyományokkal való összhang érdekében, a szerzők kérésére szerepel *döntéselőkészítés* formában.

Vákát oldal

Bevezetés

Az operáció latin szó, magyarul műveletet, műtétet jelent. Hazánkban általában kétféle értelemben használjuk. Egyrészt akkor, amikor az orvos művi úton avatkozik be valamely betegség elhárítására, másrészről pedig beszélünk harcászati hadműveleti, illetve hadászati operációkról, műveletekről.[2]

A következőkben általánosítsuk az operáció fogalmát. Nevezzünk operációnak bármely olyan beavatkozást is, amely termelési, szállítási vagy gazdasági folyamatot érint. Operációkutatás alatt pedig értsük azt a tevékenységet, amely az egyes operációk, vagyis beavatkozások legkedvezőbb kivitelezési módjait keresi.

Mivel a különféle gazdasági tevékenységeket vizsgáló operációkutatás számos matematikai, statisztikai, informatikai stb. eljárást hasznosít, a különböző tudományágak oldaláról megközelített probléma egységes szemlélete különös fontosságot kap.

A feladatok megoldása során a cél többnyire valamilyen optimum elérése. Azt az eredményt keressük, amely az adott körülmények között a legkedvezőbb. Az operációkutatás során kidolgozott többféle elképzelés jó alapot ad a vezetés számára a döntések meghozatalához.

Az operációkutatást ezek után a következőképpen definiálhatjuk: az operációkutatás a gazdasági, társadalmi tevékenységek olyan mélyreható tudományos vizsgálata, amelynek célja az optimális döntés előkészítése.[3]

Az operációkutatás lényegét az alábbiakban foglalhatjuk össze:

- a feladatot szakember(ek csoportja) végzi,
- a feladatot matematikai modell formájában fogalmazzák meg, az eredményt a modell megoldása szolgáltatja,
- tudományos módszereket alkalmaz,
- az optimális megoldásra törekedve,
- az esetek döntő többségében numerikus úton keresve a megoldást.

---

[2] Kaufmann, Arnold (1968): *Az operációkutatás módszerei és modelljei.* Budapest, Műszaki Könyvkiadó.

[3] Jándy Géza (1971): *Operációkutatás.* Budapest, Műszaki Könyvkiadó.

Gazdasági rendszerek matematikai vizsgálatára már a XIX. században is történtek kísérletek. Az első jelentősebb eredménnyel záródó kutatások azonban csak a XX. század elején indultak meg.[4]

Az operációkutatás próbaköve a második világháború volt. Ferdinand F. Leimkuhler szerint „az operációkutatás az angliai csata sötét éveiben született, hogy mozgósítsa a tudományos módszereket a nemzet fennmaradása érdekében tett kétségbeesett küzdelemhez". A Patrick Maynard Stuart Blackett vezetésével folyó munka hamarosan bizonyította az interdiszciplinaritásnak és a vizsgált problémák tudományos megközelítésének helyességét. Számos hadi, termelési, elosztási, illetve védekezési problémát oldottak meg, és az operációkutatás alkalmazása révén hatalmas költség-, anyag-, idő- stb. megtakarításokat értek el. A légi harcokban például az operációkutatás eredményeként 800 repülőgépből álló bombázóegységekben állapították meg azt a légi alakulatot, amely a legkevesebb veszteséget szenvedi. Ugyancsak az operációkutatás segítségével született az az eredmény, miszerint egy 40 hajóegységből álló és 6 hadihajó által kísért hajókonvoj az, amelynek vesztesége a legkevesebb.

A második világháború után a katonai felhasználás mellett egyre szélesebb körben megkezdődött az operációkutatás alkalmazása a polgári életben is.

Nagy lendületet adott az operációkutatás elterjedésének az egyik legáltalánosabban használt modellcsoport, a lineáris programozás elméleti és módszertani kérdéseinek a tisztázása és egyszerű számítógépre jól alkalmazható megoldásának elkészítése, amely George Bernard Dantzig (1947) nevéhez fűződik.

Az operációkutatás lényeges jellemzői közé tartozik az optimumra való törekvés. Ez a cél a közlekedésben fokozottan érvényesül. Például egy adott szállítási feladat esetén a következő célokat tűzhetjük ki magunk elé. Adott járműpark, adott típuseloszlás, adott munkaerő, adott bér, adott szállítási feladatok esetén bonyolítsuk le a feladatot a következő szempontok valamelyike(i) szerint:

- minimális távolság,
- minimális önköltség,
- maximális bevétel stb.

[4] Kaufmann, Arnold (1964): *Az optimális programozás.* Budapest, Műszaki Könyvkiadó.

Mindezekhez természetesen hozzá kell tenni azt, hogy a különböző járműtípusoknak adott viszonylatban különböző időarányos költségük van. Ugyanakkor a járművezetők részére csak az egy napra meghatározott maximális munkaidővel lehet csak számolni a tervezés során.

Mindezekből kitűnik, hogy az optimális megoldás attól függően, hogy milyen célt tűztünk ki, más és más lehet. Igen ritka az az eset, mikor mindezen célok egyidejűleg ugyanazt az optimális megoldást szolgáltatják.

Leonyid Vitaljevics Kantorovics szovjet-orosz matematikus, közgazdász már a negyvenes évek végén javasolta matematikai módszerek igénybevételét a termelés és az elosztás tervezéséhez. Ő volt az első, aki felismerte a lineáris programozási feladatok jelentőségét az ipar és a közlekedés területén. Tőle függetlenül Frank Lauren Hitchcock amerikai matematikus, fizikus és Tjalling Charles Koopmans holland–amerikai matematikus, közgazdász foglalkozott az úgynevezett szállítási problémával, ami a lineáris programozási feladatok egyik legfontosabb alkalmazási területe.

A gazdasági matematikai modellek közös jellemzője, hogy lineáris egyenletekkel és egyenlőtlenségekkel, illetve ezek kombinációival korlátozott többváltozós lineáris függvény szélsőértékeit keresik. Ezeket a szélsőértékeket egy olyan matematikai programmal kapjuk meg, ami kielégíti az előírt feltételeket, követelményeket, végrehajtása esetén pedig biztosítja az elérni kívánt gazdasági cél megvalósulását.

A lineáris programozási feladatok gyakorlati alkalmazása a második világháború idején Angliában és az Egyesült Államokban a hadseregnél kezdődött meg. A katonai alkalmazások sikerein felbuzdulva kezdték a közgazdászok ezt a módszert szélesebb területen is alkalmazni.

Vákát oldal

1. A szimplex módszer

A lineáris programozási feladatok megoldásának többféle matematikai módszere ismeretes, amelyek különböző jellegű feladatok megoldását segítik elő. Könnyen programozható és ezért általánosabban használt az úgynevezett szimplex módszer.

A lineáris programozási feladatok a matematikai értelemben vett programozásnak olyan speciális esetei, amikor a programot jellemző függvényt, valamint a programozás feltételeit lineáris matematikai kifejezések segítségével lehet megadni. Tehát a feladat nem más, mint egy lineáris függvény szélsőértékének, maximumának vagy minimumának a megkeresése. Hogy maximumot vagy minimumot keresünk, az a célfüggvénytől függ: ha az például profitot ír le, akkor maximumot, ha költséget fejez ki, akkor természetesen minimumot keresünk. Ez alapján a lineáris programozási feladatoknak két alaptípusát különböztetjük meg: a maximum- és a minimumfeladatot.[5]

A lineáris programozási feladatok lényegét és alkalmazhatóságát egy konkrét példán keresztül tudjuk szemléletesen bemutatni. Tegyük fel, hogy egy katonai üzem háromféle alapanyag felhasználásával kétféle terméket állít elő. A rendelkezésre álló alapanyag-mennyiség az „A" alapanyagból 200 egység, a „B"-ből 120 egység, a „C"-ből pedig 160 egység. Az első termék előállításához az „A" alapanyagból két egységre, a „B"-ből szintén két egységre, a „C"-ből pedig négy egységre van szükség. A második termék előállításához „A"-ból négy egységre, a „B"-ből két egységre van szükség. Itt a „C" alapanyagot nem használjuk fel. Az első termék értékesítése során a tiszta nyereség 20 €/db, a második termék esetében 30 €/db.

Annak érdekében, hogy a fent említett feladatot könnyebben át tudjuk tekinteni, célszerű az adatokat táblázatba foglalni (1.1. táblázat).

---

[5] Krekó Béla (1966): *Lineáris programozás.* Budapest, Közgazdasági és Jogi Könyvkiadó; Krekó Béla (1972): *Optimumszámítás.* Budapest, Közgazdasági és Jogi Könyvkiadó.

1.1. táblázat

alapanyag	technikai koefficiens		kapacitás
	1. termék	2. termék	
A	2	4	200
B	2	2	120
C	4	0	160
nyereség	20	30	–

Jelöljük az 1. termékből gyártandó mennyiséget x_1-gyel és a 2. termékből gyártandó mennyiséget x_2-vel. Az x_1 és az x_2 változókat döntési változóknak nevezzük. Egyértelmű, hogy az egyes termékekből gyártandó mennyiségeket úgy kell megválasztani, hogy a nyereség a lehető legnagyobb legyen. Ezt célként tűzzük ki, ezért azt az egyenletet, amely ezt matematikai formában fogalmazza meg, célfüggvénynek nevezzük:

$$C = 20x_1 + 30x_2 \quad \rightarrow \quad \max \tag{1.1}$$

Az x_1 és x_2 termékek olyan összetételét kell tehát megválasztanunk, ahol a célfüggvény szélsőértéket vesz fel. Egyértelmű az is, hogy az x_1 és x_2 negatív értéket nem vehet fel, hiszen negatív mennyiségű termelés nem lehet, tehát teljesülniük kell az alábbi feltételeknek is:

$$\begin{aligned} x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned} \tag{1.2}$$

Ha igaz az, hogy az alapanyagok felhasználása arányos a termelés mennyiségével, akkor a következő egyenlőtlenségeknek is teljesülniük kell:

$$\begin{aligned} 2x_1 + 4x_2 &\leq 200 \\ 2x_1 + 2x_2 &\leq 120 \\ 4x_1 \qquad &\leq 160 \end{aligned} \tag{1.3}$$

A felírt (1.2)–(1.3) egyenlőtlenségek a lineáris programozás feltételi egyenlőtlenség-rendszerét, az úgynevezett korlátozó feltételeket adják. A továbbiakban az egyszerűség kedvéért a változók lineárkombinációit tartalmazó oldalra „bal oldalként”, a konstans tagot tartalmazó oldalra „jobb oldalként” fogunk hivatkozni. Azokat a megoldásokat, amelyek a feltételi rendszert kielégítik, lehetséges programoknak nevezzük. A fentieket figyelembe véve

már meghatározhatjuk a lineáris programozási feladat fogalmát is. Lineáris programozási feladatnak nevezzük azokat a feladatokat, amelyeknél egy lineáris egyenlőtlenség-rendszer nemnegatív megoldásai közül egy lineáris célfüggvény szélsőértékét keressük.

A lineáris programozási feladatok matematikai szempontból igen egyszerűen kezelhetőek, és számítógéppel könnyen megoldhatóak. A továbbiakban a megoldás elvi lépéseit mutatjuk be.

Az úgynevezett szimplex módszert *(simplex method)* 1947-ben fejlesztette ki George Bernard Dantzig.[6] Egyik legnagyobb előnye, hogy felhasználható bármilyen lineáris programozási feladat megoldására. Ezenkívül előnyei közé sorolható még, hogy könnyen programozható, ezért a legtöbb matematikai, statisztikai programcsomagnak is része.

1.1. A normál feladat

A normál feladat nem más, mint egy speciális maximumfeladat, amelynek jellemzője, hogy a feltételi egyenlőtlenségekben a reláció $\leq$ a két oldal között, és a jobb oldalon kizárólag nemnegatív számok szerepelnek. A normál feladat jelentősége abban is megnyilvánul, hogy bármely lineáris programozási feladat visszavezethető egy normál feladatra. Erre a későbbiekben még visszatérünk, és konkrét példákon keresztül be is mutatjuk.

Nézzük most meg a már ismertetett feladat megoldásán keresztül, hogyan alkalmazhatjuk a normál feladatot lineáris programozási feladat megoldása esetén.

1.1.1. A normál feladat felírása szimplextáblázattal

Első lépésként készítsük el az 1.1. táblázat adatai alapján az induló táblázatunkat. A módszer sajátossága, hogy minden egyes lépés után új táblázatot kell készíteni. Ezeket a táblázatokat szimplextáblázatoknak nevezzük. Mielőtt a táblázatot elkészítenénk, meg kell ismernünk a legfontosabb jelöléseket. Az x_1, x_2 változókat már ismertettük, ezeket primális (vagy primál)

[6] Dantzig, George Bernard (1951): Maximization of a Linear Function of Variables Subject to Linear Inequalities. In Koopmans, T. C. ed.: *Activity Analysis of Production and Allocation, Cowles Commission Monograph, No. 13.* New York, Wiley.

változóknak nevezzük. Az induló program felírása érdekében vezessük be az u_1, u_2 és u_3 változókat, amelyek az egyes kapacitások fel nem használt részeit mutatják, ezeket duális (vagy duál) változóknak nevezzük. Mindezek figyelembevételével az induló programhoz az 1.2. táblázat szerinti szimplextáblázatot rendeljük.

1.2. táblázat

	x_1	x_2	
u_1	2	4	200
u_2	2	2	120
u_3	4	0	160
$-C$	20	30	0

Nézzük meg, milyen sajátosságokkal rendelkeznek a szimplextáblázatok.

- Az első oszlopban feltüntetett változók numerikus értékét mindig az utolsó oszlopban találjuk meg (programváltozók).
- Az első sorban feltüntetett változók numerikus értéke mindig nulla (szabad változók).
- A jobb alsó sarokban mindig kiolvasható a célfüggvény értéke, más szóval a program értéke, illetve a jobb alsó sarokban valójában a program értékének (–1)-szerese áll, ezért írtunk az utolsó sorban *„C”* helyett *„–C”*-t.

Vezessük be a következő jelöléseket az 1.2. táblázat alapján.

$$\mathbf{A} = \begin{bmatrix} 2 & 4 \\ 2 & 2 \\ 4 & 0 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 200 \\ 120 \\ 160 \end{bmatrix}, \quad \mathbf{c}^* = \begin{bmatrix} 20 & 30 \end{bmatrix}, \tag{1.4}$$

ahol a szimplextáblázat elemeinek mátrixát jelöljük **A**-val, a kapacitások vektorát **b**-vel, a célfüggvény együtthatóinak vektorát pedig **c**-vel.

Ekkor az (1.3) egyenlőtlenség-rendszere felírható az alábbi mátrix-egyenlettel is:

$$\mathbf{Ax} \leq \mathbf{b}, \tag{1.5}$$

továbbá a duális változók vektorára, **u**-ra igaz, hogy

(1.6) $$\mathbf{Ax} + \mathbf{u} = \mathbf{b},$$

a primális változók nemnegatív voltát megszabó (1.2) egyenlőtlenségek pedig a következő alakot veszik fel:

(1.7) $$\mathbf{x} \geq \mathbf{0}.$$

Ezzel kifejezhető a célfüggvény értéke (C) is egy skalárszorzatként, amelynek extrémumát keressük:

(1.8) $$C = \mathbf{cx} \quad \rightarrow \quad \min/\max$$

1.1.2. Dualitás

A dualitás a lineáris programozási feladatoknak azt a tulajdonságát jelenti, hogy egy adott adathalmazon két feladatot: egy maximumfeladatot és egy minimumfeladatot is értelmezhetünk. Minden lineáris programozási feladathoz (a primál feladathoz) hozzárendelhetünk egy másik lineáris programozási feladatot, amelyet az eredeti feladat duáljának nevezünk. Ha a primál feladat maximumfeladat volt, a duál minimumfeladat lesz, és fordítva.

Adott egy, az alábbi egyenletekkel jellemezhető lineáris programozási feladat, a primál feladat:

(1.9) $$\mathbf{Ax} \leq \mathbf{b},$$

(1.10) $$\mathbf{x} \geq \mathbf{0}.$$

(1.11) $$C = \mathbf{cx} \quad \rightarrow \quad \max$$

Ennek a duálja a következő lineáris programozási feladat:

(1.12) $$\mathbf{A}^{\mathrm{T}}\mathbf{y} \geq \mathbf{c},$$

(1.13) $$\mathbf{y} \geq \mathbf{0}.$$

(1.14) $$D = \mathbf{by} \quad \rightarrow \quad \min$$

Mi egy primál maximumfeladat, azaz az (1.9)–(1.11) egyenletek által leírt probléma és duáljának gazdasági jelentése? Tegyük fel, hogy a primál feladat bizonyos termékek bizonyos alapanyagokból történő előállítását írja le, ahol a profitot kell maximalizálni. Azonban nem mindegy az alapanyagok

(általánosabban: erőforrások) ára sem. Mennyit ér meg nekem az adott alapanyag? Ezt fejezi ki az (1.14) egyenlet: a szükséges alapanyagokat minél olcsóbban akarjuk beszerezni. Ugyanakkor azt is figyelembe kell venni, hogy az egyes termékek előállításához felhasznált alapanyagok összértéke nem lehet több, mint a belőlük készült termék eladási ára. Ezt fejezi ki az (1.12) egyenlet.

Mi egy primál minimumfeladat, azaz az (1.12)–(1.14) egyenletek által leírt probléma és duáljának gazdasági jelentése? Tegyük fel, hogy a primál feladat bizonyos termékek előállítását írja le különböző alapanyagokból, azaz a ráfordított költséget minimalizálnunk kell. Azonban ha a vásárló az előállított termékek egy olyan kombinációját veszi meg, hogy az egy bizonyos összetevőből a lehető legtöbbet tartalmazza, akkor ezt a feltételt az (1.11) egyenlet fejezi ki. Azonban itt is az egyes termékek előállításához felhasznált alapanyagok összértéke nem lehet több, mint a belőlük készült termék eladási ára. Ezt fejezi ki az (1.9) egyenlet.

1.1.3. A normál feladat megoldása

Az általános ismertetés után kezdjünk hozzá a feladat konkrét megoldásához.

1.1.3.1. A generálóelem kiválasztása

Első lépésként úgynevezett generálóelemet *(pivot element)* kell választani. A választás során három feltételt kell teljesíteni.

- Generálóelemet csak olyan oszlopban választhatunk, amelynek az utolsó eleme pozitív.
- A generálóelem csak pozitív szám lehet.
- Annak az oszlopnak a pozitív elemeivel, amelyből generálóelemet kívánunk választani, elosztjuk az utolsó oszlop megfelelő elemét, és az így nyert hányadosok közül kiválasztjuk a legkisebbet. A legkisebb hányados meghatározta pozitív szám lesz a generálóelem.

Vizsgáljuk meg, hogy az induló táblázatunkban melyik elemet válasszuk generálóelemnek. Látható, hogy mindkét oszlop utolsó eleme pozitív szám, így nem tudunk egyértelműen generálóelemet választani. Éppen ezért a továbbiakban célszerű generálóelemet abból az oszlopból választani, amelyben

az utolsó sor értéke a legnagyobb, azaz legnagyobb a fajlagos haszon. Példánkban tehát az x_2 oszlopból választunk generálóelemet, mert ennek az oszlopnak az utolsó eleme a legnagyobb értékű (30). Határozzuk most meg, hogy az adott oszlopban melyik lesz a generálóelem. Ehhez az utolsó oszlop elemeit elosztjuk az x_2 változó oszlopának megfelelő elemeivel:

- első sor: 200/4 = 50,
- második sor: 120/2 = 60.

A harmadik sorban nincs értelme elvégezni a számítást, mivel nulla nem lehet generálóelem.

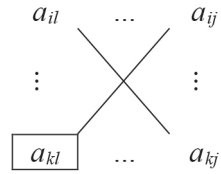
Az eredmények ismeretében már egyértelműen meghatározható a generálóelem, ami példánkban a második oszlop első eleme (4), mivel az első sorhoz tartozó hányados értéke (50) a legkisebb. A generálóelemet általában bekeretezzük a további számolások megkönnyítése érdekében.

1.1.3.2. Áttérés új szimplextáblázatra

A generálóelem meghatározása után el kell készíteni az új szimplextáblázatot. Ezt a feladatot a következő lépésekben kell végrehajtani.

1) A generálóelem sorának és oszlopának változóit felcseréljük. Tehát az új táblázatban az x_2 változó helyére u_1-et, az u_1 változó helyére pedig x_2-t írunk.
2) A generálóelem helyére beírjuk annak reciprokát. A generálóelemet jelöljük a_{kl}-lel.
3) A generálóelem sorának többi elemét megszorozzuk a generálóelem reciprokával.
4) A generálóelem oszlopának elemeit megszorozzuk a generálóelem reciprokának (–1)-szeresével.
5) Az új szimplextáblázat összes többi a_{ij}' elemét számítással, az úgynevezett téglalapszabállyal kell meghatározni a generálóelem és a kiszámítandó a_{ij}' elem sorai és oszlopai által meghatározott a_{lj} és a_{kj} elemek segítségével (lásd még 1.1. ábra):

$$a'_{ij} = a_{ij} - \frac{a_{il}a_{kj}}{a_{kl}} \tag{1.15}$$



1.1. ábra

Mindezek ismeretében már csak egyszerű számtani feladat az új táblázat elkészítése, amelynek folyamata a következő:

- Első lépésként a generálóelem sorában és oszlopában szereplő változók cserélnek helyet, vagyis az u_1 duális változó az első sorba, az x_2 primális változó helyére, az x_2 primális változó pedig az első oszlopba, az u_1 változó helyére kerül.
- Ezt követően a generálóelem helyére beírjuk annak reciprokát, tehát az a_{12} helyre 1/4-et.
- Következik a generálóelem sorának meghatározása:
  $a_{11} = 2 \quad \rightarrow \quad a_{11}' = 2/4 = 1/2$
  $b_1 = 200 \quad \rightarrow \quad b_1' = 200/4 = 50$
- Negyedik lépésben a generálóelem oszlopának elemeit határozzuk meg:
  $a_{22} = 2 \quad \rightarrow \quad a_{22}' = -2/4 = -1/2$
  $a_{32} = 0 \quad \rightarrow \quad a_{32}' = 0$
  $c_2 = 30 \quad \rightarrow \quad c_2' = -30/4 = -15/2$
- Az eddig kiszámított elemeket az 1.3. táblázat tartalmazza.

1.3. táblázat

	x_1	u_1	
x_2	1/2	1/4	50
u_2	·	–2/4	·
u_3	·	0	·
$-C$	·	–30/4	·

Ismerjük tehát már a generálóelem sorának és oszlopának új elemeit. A táblázat hiányzó elemeit a téglalapszabállyal határozzuk meg. Mielőtt azonban ehhez hozzákezdenénk, ismerkedjünk meg a téglalapszabály egy fontos tulajdonságával. Ha a generálóelem sorában vagy oszlopában (esetleg

mindkettőben) nulla értékű elem található, akkor az új szimplextáblázatban a generálóelem sorára, illetve oszlopára merőleges, a nulla elem által meghatározott oszlop, illetve sor elemei változatlanul leírhatóak. Ez abból következik, hogy ha az (1.15) kifejezésben $a_{il} = 0$ vagy $a_{kj} = 0$, akkor $a_{ij}' = a_{ij}$. Ezzel is időt takarítunk meg, és egyszerűsíthetjük a feladat megoldását.

Nézzük meg, példánkban van-e ilyen nulla elem. Látható, hogy a generálóelem oszlopának harmadik eleme nulla, tehát az oszlopra merőleges sor (a harmadik sor) elemeit az új szimplextáblázatba változatlanul leírhatjuk (1.4. táblázat).

1.4. táblázat

	x_1	u_1	
x_2	1/2	1/4	50
u_2	·	–2/4	·
u_3	4	0	160
–C	·	–30/4	·

Már csak négy elem hiányzik az új táblázat teljes kitöltéséhez, tehát a téglalapszabállyal meghatározandó elemek számát sikerült csökkenteni.

Az új táblázatra történő áttérés ötödik lépéseként határozzuk meg a hiányzó elemek értékét a téglalapszabállyal:

$a_{21} = 2 \quad \rightarrow \quad a_{21}' = 2 - 2 \cdot 2/4$

$b_2 = 120 \quad \rightarrow \quad b_2' = 120 - 2 \cdot 200/4 = 20$

$c_1 = 20 \quad \rightarrow \quad c_1' = 20 - 2 \cdot 30/4 = 5$

$-C = 0 \quad \rightarrow \quad -C' = 0 - 30 \cdot 200/4 = -1500$

Az így meghatározott elemek ismeretében ki tudjuk tölteni az új szimplextáblázatunkat (1.5. táblázat).

1.5. táblázat

	x_1	u_1	
x_2	1/2	1/4	50
u_2	1	–2/4	20
u_3	4	0	160
$-C$	5	–30/4	–1500

Mielőtt továbbmennénk, meg kell vizsgálni, hogy mi az az optimumkritérium, aminek teljesülnie kell a feladat megoldása során.

Optimális megoldást kaptunk, ha az utolsó sor valamennyi eleme (a C függvény értékének kivételével, azaz a **c** vektor összes eleme) negatív. Ekkor ez a lineáris programozási feladat egyetlen optimális megoldása.

A megoldás során alternatív optimumról beszélünk, ha a negatív számok mellett 0 is szerepel a **c** vektor elemei között.

Ha az utolsó sorban pozitív szám is van, a programot tovább kell javítani.

Mindezeket figyelembe véve láthatjuk, hogy az 1.5. táblázat még nem az optimális megoldást tartalmazza, mivel az x_1 oszlop utolsó eleme pozitív szám. Tehát az optimális eredmény elérése érdekében a programozási feladatot folytatni kell. Az új szimplextáblázatra történő áttérés ugyanúgy történik, mint azt az első lépésben már megismertük.

Generálóelemet most egyértelmű, hogy csak az x_1 oszlopban választhatunk. A generálóelem-választás harmadik feltétele szerint most a generálóelem az x_1 oszlop második eleme (1) lesz.

Az új táblázatra történő áttérés szabályait betartva az 1.6. szimplextáblázatot kapjuk.

1.6. táblázat

	x_1	u_1	
x_2	–1/2	1/2	40
u_2	1	–2/4	20
u_3	–4	2	80
$-C$	–5	–5	–1600

Látható, hogy megkaptuk az optimális megoldást, hiszen az utolsó sor elemei negatív számok.

A táblázatból kiemelve az egyes változók értékeit, a következő eredményt kaptuk:

$x_1 = 20$
$x_2 = 40$
$u_1 = 0$
$u_2 = 0$
$u_3 = 80$
$C = 1600$

Nézzük meg részletesebben, mit jelentenek és a végleges szimplextáblázatból hogyan olvashatóak le az egyes változók, illetve a célfüggvény értéke. Jelen alfejezet elején ismertettük, hogy a szimplextáblázatok milyen

sajátosságokkal rendelkeznek, vagyis az egyes változók hol szerepelnek a táblázatban. Azt mondtuk, hogy az első sorban szereplő változók értéke mindig nulla. Az induló táblázatban az x_1 és x_2 primális változók szerepeltek az első sorban. A feladat megoldása során azonban ezek a primális változók helyet cseréltek az u_1 és u_2 duális változókkal, így u_1 és u_2 értéke lett nulla.

Az első oszlopban szereplő változók (függetlenül attól, hogy primális vagy duális változók) értékeit az utolsó oszlopban olvashatjuk le. Tehát az első oszlopban szereplő primális változók értéke: $x_1 = 20$, $x_2 = 40$, valamint szintén az első oszlopban szereplő duális változó, u_3 értéke 80 lesz.

Az optimális program értékét mindig a táblázat jobb alsó sarkában (utolsó sor utolsó eleme) olvashatjuk le. Mint említettük, itt a célfüggvény értékének (–1)-szerese található, tehát a megoldásnál ellentétes előjellel vesszük figyelembe: $C = 1600$.

A feladat kezdetén a problémát szövegesen adtuk meg, célszerű tehát a jobb áttekinthetőség és megérthetőség kedvéért az eredményt is szöveges formába önteni, ami példánkban a következő lehet:

A katonai üzem optimális termelés esetén az első termékből 20 egységet, a másodikból 40 egységet állít elő. A termelés során a rendelkezésre álló alapanyagok közül az első és második csoportba tartozó alapanyagokat teljes mértékben felhasználja, a harmadik csoportba tartozóból azonban marad 80 egység. A gyártó az összes végterméket értékesíteni tudta a tervben szereplő nyereséghányaddal, így összesen 1600 € tiszta nyereségre tett szert: $20 \cdot 20 + 30 \cdot 40 = 1600$.

Végezetül meg kell még jegyezni, hogy ha az utolsó sor elemei között van pozitív szám, de a hozzá tartozó oszlopban nem találunk pozitív számot, azaz nem tudunk generálóelemet választani, a feladatnak nincs megoldása.

1.2. Módosított normál feladat

A normál feladatnál a feltételi egyenlőtlenségek csak kisebb vagy egyenlő ($\leq$) szimbólummal szerepeltek. A módosított normál feladatnál azonban feltételezzük, hogy a feltételek között van egyenlőség is, vagy bizonyos esetekben csak egyenlőségek szerepelnek. Nézzük meg, hogyan jelentkezik ez a probléma egy konkrét példában.

Egy laktanyából kétfajta gépjárműtechnikai anyagot kell helyőrségen kívülre szállítani, és ehhez tehergépjárműveket kell igénybe venni. Tegyük fel, hogy a bérbeadó három különböző gépjárművet tud csak az adott időpontban

kiállítani. Az első gépkocsira az „A” termékből 2 db, a „B” termékből 3 db, a második gépkocsira 2 és 5 db, a harmadikra 1 és 4 db termék fér fel egyszerre. Az „A” termékből 88 db-ot, a másodikból 200 db-ot kell elszállítani. A szállítás fajlagos költsége fordulónként az alábbi:

- az első gépjárművel 4000 Ft/forduló
- a második gépjárművel 6000 Ft/forduló
- a harmadik gépjárművel 2000 Ft/forduló

Hány fordulót végezzünk az egyes gépjárművekkel, ha minimális költséget akarunk elérni, és mekkora lesz ez a minimális költség?

Első lépésként próbáljuk meg megfogalmazni a feladatot matematikai formában és felírni a lineáris programozási feladat feltételeinek egyenlőtlenségi (egyenlőségi) rendszerét. Példánkban a primális változók a szükséges gépjárművek lesznek, vagyis az első gépjárműből szükséges darabszámot x_1-gyel, a másodikat x_2-vel, a harmadikat x_3-mal jelöljük. A duális változók az elszállítandó árumennyiségek, így az u_1 változó értéke 88, az u_2 változóé pedig 200.

A célfüggvény felírása sem okoz problémát, mivel azt az egyes gépjárművekkel teljesített fordulók számának és a szállítás fajlagos költségének szorzata adja. Érdekessége a feladatnak, hogy a célfüggvényt most minimumfeladatra kell felírni. A feladat megoldása során ez annyiban jelent változást, hogy a célfüggvényt (–1)-gyel meg kell szorozni, és az így nyert értékeket kell az induló táblázat utolsó sorába beírni.

1.2.1. A módosított normál feladat szimplextáblázatának felírása

A fentiek figyelembevételével már matematikai formában is felírható az egyenlőtlenség-rendszer:

$$x_1 \geq 0 \qquad x_2 \geq 0 \qquad x_3 \geq 0 \tag{1.16}$$

$$2x_1 + 2x_2 + x_3 = 88 \qquad 3x_1 + 5x_2 + 4x_3 = 200 \tag{1.17}$$

(1.18) $$C = 4000x_1 + 6000x_2 + 2000x_3 \rightarrow \min$$

Az alapadatok ismeretében elkészíthető az induló szimplextáblázat is, amelyet az 1.7. táblázatban láthatunk.

1.7. táblázat

	x_1	x_2	x_3	
u_1	2	2	1	88
u_2	3	5	4	200
C	–4000	–6000	–2000	0

Ha az 1.7. táblázatból kiindulva, a normál feladatnál megismert módon oldanánk meg a feladatot, akkor az optimális programban az u_1 és u_2 értéke akár pozitív is lehetne. Ebben az esetben a feladat (1.17) szerinti feltételei nem teljesülnének egyenlőség formájában. Márpedig a feladat érdekében minden körülmények között biztosítani kell, hogy az u_1 és u_2 értékek nullává váljanak. Ezt a következőképpen érhetjük el.

Abból indulunk ki, hogy egy duális változó mindig olyan nemnegatív számot jelent, amely megmutatja, hogy a megfelelő egyenlőtlenség bal oldala mennyivel kisebb a jobb oldalnál. Ezt figyelembe véve:

$$u_1 = 88 - (2x_1 + 2x_2 + x_3)$$
$$u_2 = 200 - (3x_1 + 5x_2 + 4x_3)$$

A két duális változó összege tehát a következőképpen írható fel:

$$u_1 + u_2 = 288 - (5x_1 + 7x_2 + 5x_3)$$

Mivel sem az u_1, sem az u_2 értéke nem lehet negatív, egyértelmű, hogy az

$$5x_1 + 7x_2 + 5x_3$$

kifejezés maximális értéke 288. Ha ez a kifejezés esetleg eléri ezt az értéket, akkor mind az u_1, mind az u_2 értéke nullává válik. Ellenkező esetben ez nem következik be.

Ezt a problémát úgy oldjuk meg, hogy az eredeti célfüggvény mellett átmenetileg az

$$5x_1 + 7x_2 + 5x_3$$

kifejezést is felvesszük másodlagos célfüggvényként. Első lépésben ennek a másodlagos célfüggvénynek a maximumát keressük meg. Ha ez a maximum

kisebb 288-nál, a feladatnak nincs megoldása. Azonban ha a másodlagos célfüggvény maximuma 288, akkor mind az u_1, mind az u_2 értéke nulla lesz. Mindezeket figyelembe véve az induló szimplextáblázatunk az 1.8. táblázat szerint módosul.

A táblázat annyiban változik, hogy a másodlagos célfüggvényt a táblázat utolsó soraként tüntetjük fel. Azokat a duális változókat, melyeknek nullává kell válni, csillaggal jelöltük meg.

1.8. táblázat

	x_1	x_2	x_3	
u_1^*	2	2	1	88
u_2^*	3	5	4	200
C	–4000	–6000	–2000	0
	5	7	5	288

Az 1.8. táblázat alapján a feladat már megoldható a normál feladatnál megismert módszerrel. Mivel ezzel az eljárással először a másodlagos célfüggvény maximumát határozzuk meg, és csak ezt követően az eredeti célfüggvény minimumát, a megoldás hosszabb és bonyolultabb. Éppen ezért a gyakorlatban ezt ritkán használják, így a megoldás részletes ismertetésével itt nem foglalkozunk.

1.2.2. Egyszerűsített megoldás

Ismert azonban a módosított normál feladat megoldására egy egyszerűbb, a gyakorlatban is jól alkalmazható eljárás. A másodlagos célfüggvény együtthatóit nem írjuk fel, mivel ezeket az együtthatókat bármikor előállíthatjuk, ha a csillagos sorok elemeit oszloponként összegezzük. Írjuk tehát fel az egyszerűsített megoldás induló szimplextáblázatát (1.9. táblázat).

1.9. táblázat

	x_1	x_2	x_3	
u_1^*	2	2	1	88
u_2^*	3	5	[4]	200
C	–4	–6	–2	0

A feladat megoldása során az a célunk, hogy a csillagos változók az első sorba kerüljenek. Ennek érdekében generálóelemet abban az oszlopban választunk, amelyben a csillagos sorok együtthatóinak összege pozitív. Ha csillagos változó még marad az első oszlopban, de a csillagos sorok oszloponkénti összegzésével már nem kapnánk pozitív számot, vagyis nem tudnánk generálóelemet választani, akkor a feladatnak nincs megoldása. Abban az esetben viszont, ha a csillagos sorok összegzésénél nullát kapunk, akkor az ennek megfelelő oszlopban folytathatjuk a számításokat.

A fenti alapismeretek után kezdjünk hozzá a módosított normál feladat egyszerűsített módszerrel történő megoldásához. Első lépésként a generálóelemet kell most is kiválasztani.

Először vizsgáljuk meg, hogy a csillagos sorok együtthatóinak összege az egyes oszlopokban hogyan alakul:

első oszlop: 2 + 3 = 5
második oszlop: 2 + 5 = 7
harmadik oszlop: 1 + 4 = 5

Látható, hogy a csillagos sorok együtthatójának oszloponkénti összege mindenhol pozitív. Így generálóelemet most abban az oszlopban választunk, ahol az utolsó sorbeli érték abszolút értékben a legkisebb, minimumfeladatról lévén szó. Ezt figyelembe véve példánkban az 1.9. táblázat utolsó sorának abszolút értékben legkisebb eleme –2, ami egyben a harmadik oszlop utolsó eleme is. Ez azt jelenti, hogy generálóelemet az x_3 oszlopban kell választani. A generálóelem-választás első két szabálya nem határozza meg egyértelműen a generálóelem helyét, így a harmadik szabály alapján járunk el. Ez azt jelenti, hogy a harmadik oszlop elemeivel elosztjuk az utolsó oszlop megfelelő elemeit, és az így kapott hányadosok közül a legkisebb határozza meg a generálóelem helyét:

első sor: 88/1 = 88
második sor: 200/4 = 50.

Tehát a generálóelem a második sor eleme (4) lesz (1.9. táblázat).

Megfigyelhető még az induló táblázatban az is, hogy az utolsó sor, azaz a célfüggvény elemeit az egyszerűbb számítás érdekében 100-zal elosztottuk. Arra kell csak ügyelni, hogy a feladat megoldása után a célfüggvény értékét ugyanezzel a számmal beszorozzuk, hogy a valós eredményt kapjuk meg.

Az induló szimplextáblázatról az új táblázatra a már megismert módon térünk át, annyi módosítással, hogy valahányszor csillagos sorban választunk

generálóelemet, a neki megfelelő oszlopot a következő táblázatból elhagyjuk, azaz csillagos változó oszlopát egyszerűen nem írjuk le. A csillagos változó első sorba történt bevitelével ugyanis elértük, hogy értéke nulla legyen, ami az egyenlőségi feltételéhez szükséges, ezért ezután már nem akarjuk visszavinni az első oszlopba.

Mindezeket figyelembe véve az új szimplextáblázatot a következő szerint készítjük el:

- az x_3 változó és a második sor csillagos változója helyet cserél,
- az első sorba került csillagos változóhoz tartozó oszlopot nem írjuk le, tehát az új táblázatban a generálóelem transzformált értékét, illetve a generálóelem oszlopát nem kell számítani,
- a generálóelem sorában az új elemek értékét a normál feladatnál bemutatott módon határozzuk meg.

Az eddig ismertetett lépéseket végrehajtva az 1.10. táblázatot kapjuk.

1.10. táblázat

	x_1	x_2	
u_1^*	·	·	·
x_3	3/4	5/4	50
C	·	·	·

A táblázat hiányzó elemeit a téglalapszabállyal határozzuk meg:

$a_{11} = 2 \quad \rightarrow \quad a_{11}' = 2 - 1 \cdot 3/4 = 5/4$

$a_{12} = 2 \quad \rightarrow \quad a_{12}' = 2 - 1 \cdot 5/4 = 3/4$

⋮

A még hiányzó elemek meghatározását tovább nem részletezzük, hiszen a meghatározás módszere azonos az a_{11} és a_{12} elemeknél bemutatottakkal. A téglalapszabály ismételt végrehajtása után már teljes mértékben kitölthető az új szimplextáblázat (1.11. táblázat).

Az új táblázatban még van csillagos sor, ezért ebben kell generálóelemet választani. Látható, hogy a csillagos sorban mindkét érték pozitív, vagyis nem tudunk egyértelműen generálóelemet választani. Tehát most is az utolsó sor abszolút értékben legkisebb eleme határozza meg a generálóelem oszlopát. Eszerint az x_1 oszlop első eleme (5/4) lesz a generálóelem. Az új táblázatra történő áttérés szintén a már megismert módon történik (1.12. táblázat).

1.11. táblázat

	x_1	x_2	
u_1^*	5/4	3/4	38
x_3	3/4	5/4	50
C	-10/4	-14/4	100

1.12. táblázat

	x_2	
x_1	3/5	152/5
x_3	4/5	136/5
C	-2	176

Amennyiben a csillagos változókat sikerült az első sorba juttatnunk, a továbbiakban az optimumkritérium a normál feladatéval azonos. Példánkban is ez az eset áll fenn. Az optimumkritériumot figyelembe véve látható, hogy optimális megoldáshoz jutottunk, hiszen az utolsó sorban csak negatív szám szerepel.

Ha még maradt volna az első oszlopban csillagos változó, de már nem tudnánk generálóelemet választani, akkor a feladatnak nincs megoldása. Olyan eset is előfordulhat, hogy a csillagos változók első sorba juttatása után az optimumkritérium még nem teljesül. Ekkor a feladatot normál feladat szerint kell tovább folytatni, mindaddig, amíg a feltételek nem teljesülnek.

Nézzük most meg, hogy a feladatnak az egyszerűsített módszerrel történt megoldása milyen eredményt hozott.

$x_1 = 152/5 = 30{,}4$

$x_2 = 0$

$x_3 = 136/5 = 27{,}2$

$C = 176 \cdot 1000 = 152/5 \cdot 4000 + 136/5 \cdot 2000 = 121\,600 + 54\,400 = 176\,000.$

Összefoglalva megállapíthatjuk, hogy az első gépkocsival 30,4 fordulót, a harmadik gépkocsival pedig 27,2 fordulót kell teljesíteni. A második gépjárművet nem vesszük igénybe a szállítási feladat során. Az első és harmadik gépjárművel az összes anyagot elszállítottuk. A minimális szállítási költség 176 000 Ft lett.

Egyértelmű, hogy a gyakorlatban nem töredék, hanem egész fordulókkal kell számolni. Ezt figyelembe véve 31, illetve 28 fordulót kell teljesíteni, azaz a teljes szállítási költség $31 \cdot 4000 + 28 \cdot 2000 = 180\,000$ Ft lesz.

1.3. Általános feladat

Próbáljuk meg megoldani szimplex programozással a következő feladatot.

Egy gazdaságban minden szarvasmarhának legalább 6 egység fehérjét, 12 egység szénhidrátot és 4 egység zsírt kell kapnia naponta.

Az etetéshez kétféle takarmányt használnak fel. Az egyes takarmányfélék tápanyagtartalmát egységnyi tömegre vonatkoztatva az 1.13. táblázat mutatja.

1.13. táblázat

tápanyag	1. takarmány (x_1)	2. takarmány (x_2)
fehérje	2	4
szénhidrát	2	2
zsír	4	0

Milyen takarmány-összetételnél lesz az önköltség minimális, ha az egyes takarmányfajták önköltsége 5, illetve 6 €?

Első lépésként értelmezni kell a feladatot, és felírni a feltételi egyenlőtlenségeket, majd a célfüggvényt:

(1.19)
$$x_1 \geq 0$$
$$x_2 \geq 0$$

(1.20)
$$2x_1 + x_2 \geq 6$$
$$2x_1 + 4x_2 \geq 12$$
$$4x_2 \geq 4$$

(1.21)
$$C = 5x_1 + 6x_2 \rightarrow \min$$

Látható, hogy ebben az esetben, ellentétben a normál feladatnál (≤), illetve a módosított normál feladatnál (=) megismert feltételi egyenlőtlenségekkel (egyenlőségekkel), az (1.20) feltételben ≥ relációjelek is szerepelhetnek, esetleg kizárólag ilyenek jelennek meg. Egyértelmű, hogy így a feladatot a már bemutatott módszerek egyikével sem lehet megoldani.

1.3.1. Az általános feladat szimplextáblázatának felírása

A megoldás első lépéseként alakítsuk át a feltételi egyenlőtlenségeket egyenlőségekké, mégpedig úgy, hogy új változókat vezetünk be. Jelentsék ezek a változók azt a nemnegatív számot, amely megmutatja, hogy az (1.20) egyenlőtlenségekben szereplő feltételeknél az x_1 és x_2 primális változók összege mennyivel nagyobb az egyenlőtlenség jobb oldalán lévő számnál. Példánkban ezen változók értéke az alábbiak szerint írható fel:

$$\begin{aligned} v_1 &= 2x_1 + x_2 - 6 \\ v_2 &= 2x_1 + 4x_2 - 12 \\ v_3 &= 4x_2 - 4 \end{aligned} \tag{1.22}$$

Az új változók bevezetésével az (1.20) egyenlőtlenségei helyett egyenlőségeket írhatunk fel. Ezzel együtt persze a célfüggvényünket is módosítani kell. Ez úgy történik, hogy az eredeti célfüggvényt kiegészítjük a $0 \cdot v_1$, $0 \cdot v_2$, $0 \cdot v_3$ tagokkal. Mindezeket figyelembe véve már fel tudjuk írni a módosított feladatra a feltételi rendszert:

$$\begin{aligned} x_1 &\geq 0 \\ x_2 &\geq 0 \\ v_1 &\geq 0 \\ v_2 &\geq 0 \\ v_3 &\geq 0 \end{aligned} \tag{1.23}$$

$$\begin{aligned} 2x_1 + x_2 - v_1 &= 6 \\ 2x_1 + 4x_2 - v_2 &= 12 \\ 4x_2 - v_3 &= 4 \end{aligned} \tag{1.24}$$

$$C = 5x_1 + 6x_2 + 0v_1 + 0v_2 + 0v_3 \quad \rightarrow \quad \min \tag{1.25}$$

Látható, hogy eredeti feladatunkat az új változók segítségével módosított normál feladatra vezettük vissza. Így a feladat az ott megismert módszerrel már megoldható. Az egyszerűsített eljárást figyelembe véve induló táblázatunkat az 1.14. táblázat szerint kell elkészíteni.

1.14. táblázat

	x_1	x_2	v_1	v_2	v_3	
u_1^*	2	1	–1	0	0	6
u_2^*	2	4	0	–1	0	12
u_3^*	0	4	0	0	–1	4
C	–5	–6	0	0	0	0

Mivel minimumfeladatot oldunk meg, az induló szimplextáblázat utolsó sorába nem az eredeti célfüggvényt, hanem annak (–1)-szeresét írjuk be. Vagyis a minimumfeladat helyett az eredeti célfüggvény (–1)-szeresének maximumát fogjuk keresni.

Bemutatott példánk módosított normál feladatra történő visszavezetése az induló szimplextáblázat (1.14. táblázat) jelentős bővülésével járt együtt. Általános érvénnyel megállapítható, hogy minél több ≥ jellegű feltétel van a feltételi egyenlőtlenségekben, az induló táblázatunk azzal egyenes arányban bővül. Ez oda vezet, hogy bonyolultabb feladatok esetén a számítás igen hosszadalmassá válik. Éppen ezért a gyakorlatban nem célszerű ezt a módszert alkalmazni.

Kézenfekvő, hogy itt is (a módosított normál feladathoz hasonlóan) keressünk egy egyszerűsített megoldást.

1.3.2. Egyszerűsített megoldás

Az egyszerűsített megoldás lényege, hogy a v_i változókat nem a primális, hanem a duális változók között szerepeltetjük, negatív előjellel. Ez azt jelenti, hogy a megfelelő egyenlőtlenségeknél nem a szokásos ≤ reláció, hanem az ellenkező értelmű ≥ reláció van érvényben. A $(-v_i)$ változókhoz tartozó sorokat negatív soroknak fogjuk nevezni. Ezzel sikerült azt elérni, hogy az induló szimplextáblázatunkat egyáltalán nem kell bővíteni (1.15. táblázat).

1.15. táblázat

	x_1	x_2	
$-v_1$	2	1	6
$-v_2$	2	4	12
$-v_3$	0	4	4
C	–5	–6	0

Az egyszerűsített feladat megoldása során célunk az, hogy a csillagos (ha van) és a negatív sorokat ki tudjuk küszöbölni, vagyis a negatív és a csillagos változókat az első sorba kell juttatni. Ennek megfelelően generálóelemet mindig olyan oszlopban választunk, ahol a csillagos és a negatív sorokhoz tartozó elemek együttes összege pozitív szám. Ha csillagos sorból kerül ki a generálóelem, akkor a következő táblázatból a megfelelő oszlop kiesik. Más a helyzet, ha negatív sorban van a generálóelem. Ilyen esetben az új táblázatból nem esik ki az oszlop, erre vonatkozóan is el kell végezni a számításokat. A számítások végrehajtása után a kérdéses oszlop minden egyes elemét meg kell szorozni (–1)-gyel. Ez azt eredményezi, hogy az adott ($-v_i$) változó pozitívvá válik.

Mindezek figyelembevételével kezdjük el példánk megoldását. Egyszerű helyzetben vagyunk, hiszen az 1.15. táblázatban csak negatív sorok vannak. Mindkét oszlopban a negatív sorokhoz tartozó elemek összege pozitív, így generálóelemet most a módosított normál feladatnál bemutatott módon kell választani. Példánkban a generálóelemet az x_1 oszlopban választjuk, mert abszolút értékben ennek az oszlopnak az értéke a legkisebb. Nézzük meg, hogy az adott oszlopban melyik elem lesz a generálóelem:

első sor: 6/2 = 3

második sor: 12/2 = 6.

A harmadik sorban szükségtelen a számítás elvégzése, mert nulla értékű elemet nem választhatunk generálóelemnek. A kapott eredmény azt mutatja, hogy az első sor eleme lesz a generálóelem (2), mivel itt a legkisebb a hányados értéke.

Ezt követően el kell készíteni az új szimplextáblázatunkat. Először a változók cserélnek helyet, vagyis az x_1 változó az első sorba kerül, a $-v_1$ változó pedig az első oszlopba, de már pozitív előjellel. Az új táblázat elemeinek meghatározása a már ismertetett módon történik, annyi módosítással, hogy a generálóelem oszlopának minden elemét megszorozzuk (–1)-gyel.

Végezzük el az új táblázat elkészítéséhez szükséges számításokat.

- A generálóelem új értékének meghatározása:
  $a_{11} = 2 \quad \rightarrow \quad a_{11}' = -1/2$
- A generálóelem oszlopa elemeinek meghatározása:
  $a_{21} = 2 \quad \rightarrow \quad a_{21}' = 2/2 = 1$
  $a_{31} = 0 \quad \rightarrow \quad a_{31}' = 0/2 = 0$
  $c_1 = -5 \quad \rightarrow \quad c_1' = -5/2$

- A generálóelem sora elemeinek meghatározása:
  $a_{12} = 1 \quad \rightarrow \quad a_{21}' = 1/2$
  $b_1 = 6 \quad \rightarrow \quad b_1' = 6/2 = 3$

A fenti számítások eredményeit már beírhatjuk az új táblázatba (1.16. táblázat).

Figyeljünk arra, hogy a generálóelem oszlopában van nulla, a harmadik elem, így a rá merőleges sor, a harmadik sor elemeit változatlanul beírhatjuk az új szimplextáblázatba (1.17. táblázat):

1.16. táblázat

	v_1	x_2	
x_1	–1/2	1/2	3
$-v_2$	1	·	·
$-v_3$	0	·	·
C	–5/2	·	·

1.17. táblázat

	v_1	x_2	
x_1	–1/2	1/2	3
$-v_2$	1	·	·
$-v_3$	0	4	4
C	–5/2	·	·

Így a téglalapszabállyal már csak négy elemet kell kiszámítani, amit a normál feladatnál megismert és részletesen bemutatott módon hajtunk végre. A számítások elvégzése után az 1.18. táblázathoz jutunk.

Folytassuk a feladatot a fent bemutatott módon, hiszen még van negatív sor. Az új generálóelemet a v_1 oszlopban kell választani. Egyértelmű, hogy csak a második sor eleme (1) lehet generálóelem.

Mielőtt továbbmennénk, meg kell ismerni egy fontos szabályt. Ha egy negatív változó az első sorba került, és ott pozitív előjelűvé vált, a feladat folytatása során pozitív előjelű változóként visszakerülhet az első oszlopba.

Példánkban ez a szituáció áll most fenn, tehát a feladatot nyugodtan folytathatjuk. A számítások elvégzése után az 1.19. táblázatot kapjuk.

1.18. táblázat

	v_1	x_2	
x_1	–1/2	1/2	3
$-v_2$	[1]	3	6
$-v_3$	0	4	4
C	–5/2	–7/2	15

1.19. táblázat

	v_2	x_2	
x_1	–1/2	2	6
v_1	–1	3	6
$-v_3$	0	[4]	4
C	–5/2	4	30

Még mindig van negatív sor a táblázatban, így az előzőek szerint folytatjuk a feladat megoldását. Generálóelemet csak a második oszlopban

választhatunk, és ott csak a harmadik sor eleme lehet az (4). Elvégezve a változók cseréjét és a számításokat, az 1.20. táblázatot kapjuk.

Sikerült minden negatív sort kiküszöbölni az ismertetett módszerrel. Ha ez nem sikerült volna, akkor a feladatnak nincs megoldása, hiszen nincs egyetlen realizálható program sem. Példánkban a számítások első fázisát (a negatív és üres sorok kiküszöbölése) befejeztük. Ezzel azonban még a feladatot nem oldottuk meg. Ha figyelmesen megnézzük az 1.20. táblázatot, látható, hogy a negatív sorok kiküszöbölésével egy normál feladat szimplextáblázatához jutottunk. Normál feladat esetén abban az oszlopban kell generálóelemet választani, amelyiknek utolsó eleme pozitív szám. Ez a feltétel a második oszlop esetén fennáll. Tehát ha a generálóelem választásának normál feladatnál megismert feltételei teljesülnek, a feladatot normál feladatként kell tovább folytatni. Esetünkben a második sor második eleme lesz a generálóelem (3/4). Az új szimplextáblázatra történő áttérés is a normál feladatnál tanult módon történik, így a számítások elvégzése után az 1.21. táblázatot kapjuk.

1.20. táblázat

	v_2	v_3	
x_1	–1/2	2/4	4
v_1	–1	**3/4**	3
x_2	0	–1/4	1
C	–5/2	1	26

1.21. táblázat

	v_2	v_1	
x_1	1/6	–2/3	2
v_3	–4/3	4/3	4
x_2	–1/3	1/3	2
C	–7/6	–4/3	22

Mivel az utolsó sor minden eleme (kivéve a célfüggvény értékét) negatív, így eljutottunk az optimális megoldáshoz. A kapott eredmény a következő lesz:

$x_1 = 2$
$x_2 = 2$
$x_3 = 0$
$C = 2 \cdot 5 + 2 \cdot 6 = 22.$

Szövegesen megfogalmazva ez azt jelenti, hogy az állatok etetéséhez az x_1 takarmányból és az x_2 takarmányból egyaránt két egységet használunk fel, így egy szarvasmarha etetése naponta 22 €-ba kerül.

A normál, a módosított normál és az általános feladatnál bemutatott módszerekkel bármilyen lineáris programozási feladatot meg lehet oldani, ha a feladatnak az adott feltételek mellett van megoldása. Ahhoz azonban,

hogy a bemutatott módszereket a gyakorlatban is alkalmazni tudjuk, úgy kell átfogalmazni a feladatokat, hogy azok a következő feltételeknek eleget tegyenek:

- a feltételi egyenlőtlenségek jobb oldalán nemnegatív szám szerepeljen;
- az optimumot a célfüggvény maximuma jelentse.

2. A szállítási probléma

2.1. A szállítási probléma megfogalmazása

Ebben a fejezetben a lineáris programozás egyik speciális ágával, az úgynevezett szállítási problémával foglalkozunk. Először az egyszerű szállítási problémát ismertetjük. Ez azonban, mint látni fogjuk, a gyakorlatot igen leegyszerűsíti, ennélfogva nagyrészt nem alkalmas a gyakorlatban jelentkező számos feladat megoldására. Éppen ezért foglalkoznunk kell az általános szállítási probléma megfogalmazásával és megoldásával is.

A következőkben fogalmazzuk meg az egyszerű szállítási problémát. Tegyük fel, hogy valamely városban, országban, egyszóval egy területegységen kibocsátóhelyek állnak rendelkezésre. Jelöljük ezeket K_1, K_2,… K_m-mel. Ugyanakkor fogadóhelyek is rendelkezésre állnak, amelyeket jelöljünk F_1, F_2,…, F_n-nel.

Általánosságban kibocsátóhelynek nevezhetünk minden olyan gyártóvagy más egységet, ahonnan árut lehet elszállítani, fogadóhelynek pedig minden olyan raktárat, üzletet, felhasználót stb., ami árut igényel.

Írjunk fel egy konkrét példát, és ezen keresztül fogalmazzuk meg a problémát. Tegyük fel, hogy egy városban öt sétálóutcát kell felújítani. A munkálatokhoz szükséges díszkövet négy telephelyről lehet szállítani. A telephelyek egyenként 40, 70, 60 és 30 kocsirakománynak megfelelő díszkőmennyiséget tudnak rendelkezésre bocsátani. Az egyes építési helyeknek (fogadóhelyeknek) az igénye 30, 60, 50, 40 és 20 kocsirakomány. Lényeges megjegyezni, hogy azonos típusú (teherbírású) kocsikat feltételezünk a feladat végrehajtásánál. A feladat megoldásához ismernünk kell a fogadó- és kibocsátóhelyek távolságát is, amely adatokat a 2.1. táblázat tartalmazza, kiegészítve az igények és a kapacitások értékeivel.

2.1. táblázat

kibocsátóhelyek	F_1	F_2	F_3	F_4	F_5	kapacitás
K_1	6	2	8	7	5	40
K_2	4	3	7	5	9	70
K_3	2	1	3	6	4	60
K_4	5	6	4	8	3	30
szükséglet	30	60	50	40	20	200

A táblázat az alábbi elemekből épül fel:

- az első sor a fogadóhelyeket,
- az utolsó sor a fogadóhelyek szükségletét (**b**),
- az első oszlop a kibocsátóhelyeket,
- az utolsó oszlop a kibocsátóhelyek kínálatát (**a**),
- a táblázat belseje pedig az egyes kibocsátóhelyek és fogadóhelyek egymás közötti távolságát tünteti fel (**C**).

Itt kell megjegyezni, hogy a táblázatba nem feltétlenül szükséges távolságértékeket írni. Előfordulhat, hogy az átszállítás egységköltségeit, esetleg az elérhető nyereségeket vesszük figyelembe a feladat megoldásánál. Ilyenkor persze nem a minimális futás, hanem a minimális költség vagy a maximális nyereség elérése a célunk. Ennek megfelelően beszélhetünk költségtáblázatról, vagy nyereségmátrixról, esetleg teljesítménymátrixról stb.

A fentiek alapján már meg tudjuk fogalmazni, mit is értünk szállítási probléma alatt: adott kibocsátóhelyekről adott fogadóhelyekre történő szállítás valamilyen szempont (minimális futás, minimális költség, maximális nyereség stb.) szerinti optimalizálása.

A probléma megoldásának számtalan változata van. Ezek közül kell kiválasztani a legjobbat. Próbáljuk meg ezt most általánosságban megfogalmazni. Ennek érdekében az előbb említett táblázatot, illetve mátrixot (függetlenül annak tartalmától, vagyis hogy teljesítményt, költséget stb. reprezentál) nevezzük célmátrixnak vagy költségmátrixnak. Ha a célt optimumkritériumnak nevezzük, akkor a célmátrixunk fejezi ki azt, hogy tulajdonképpen a feladatot mi célból akarjuk megoldani.

Az optimális megoldás tartalmazza azokat az x_{ij} értékeket, amelyek azt jelentik, hogy az *i*-edik kibocsátóhelyről a *j*-edik fogadóhelyre mennyi díszkövet kell szállítani. Ezeket az értékeket szintén mátrix formájában

lehet felírni. Ezt a mátrixot diszpozíciós mátrixnak nevezzük, és **X**-szel jelöljük (2.2. mátrix).

2.2. mátrix

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \\ x_{41} & x_{42} & x_{43} & x_{44} & x_{45} \end{bmatrix}$$

A feladat megoldása során láthatjuk majd, hogy a mátrix elemei közül (az x_{ij} értékek közül) bizonyos számúnak nullával kell egyenlőnek lennie, ha a feladatot optimálisan oldottuk meg. Ezen túlmenően az is igaz, hogy a többi elem csak pozitív szám lehet. Tehát összességében a mátrix elemei csak nemnegatív számok lehetnek, vagyis:

(2.1) $$x_{ij} \geq 0 \quad \forall i, j.$$

A feladat megoldásánál célul tűztük ki, hogy a gépkocsik összes futása minimális legyen. Ennek érdekében az alábbi függvényt, az úgynevezett célfüggvényt írhatjuk fel:

(2.2) $$C = 6x_{11} + 2x_{12} + \ldots + 5x_{15} + 4x_{21} + \ldots + 3x_{45} \quad \rightarrow \quad \min$$

Láthatjuk, hogy a célmátrix és a diszpozíciós mátrix elemeit szoroztuk össze, majd ezeket összeadva keletkezett a célfüggvény. Ezt általánosan megfogalmazva, egyszerűbb matematikai jelöléssel az alábbiak szerint fejezhetjük ki:

(2.3) $$C = \sum_{i,j} c_{ij} x_{ij} \quad \rightarrow \quad \min$$

Két azonos dimenziójú mátrixnak ezt a fajta, elemenkénti szorzatát (a valódi mátrixszorzástól való alapvető különbözősége miatt) más névvel illetjük, ez az úgynevezett Schur-szorzat vagy Hadamard-szorzat. Jelölése: $\mathbf{C} \circ \mathbf{X}$.

A célfüggvény nem minden x_{ij} esetben érvényes, vannak korlátozó feltételek is, amelyek a következők:

(2.4)
$$\begin{aligned} x_{11} + x_{12} + x_{13} + x_{14} + x_{15} &= 40 \\ x_{21} + \ldots + x_{25} &= 70 \\ x_{31} + \ldots + x_{35} &= 60 \\ x_{41} + \ldots + x_{45} &= 30 \end{aligned}$$

(2.5)
$$\begin{aligned} x_{11} + x_{21} + x_{31} + x_{41} &= 30 \\ x_{12} + \ldots + x_{42} &= 60 \\ x_{13} + \ldots + x_{43} &= 50 \\ x_{14} + \ldots + x_{44} &= 40 \\ x_{15} + \ldots + x_{45} &= 20 \end{aligned}$$

Egyszerűbb matematikai jelöléssel az előzőek a következőképpen írhatók le:

(2.6)
$$\sum_{j=1}^{n} x_{ij} = a_i ,$$

(2.7)
$$\sum_{i=1}^{m} x_{ij} = b_j ,$$

ahol a_i az *i*-edik kibocsátóhely kapacitása, b_j pedig a *j*-edik fogadóhely igénye. Az első egyenlet a K_i kibocsátóhelyről elszállítható mennyiséget mutatja, míg a második egyenlet azt fejezi ki, hogy az F_j fogadóhelyre nem lehet annak igényénél több árut (de kevesebbet sem) szállítani.

Végül ismerkedjünk meg néhány igen fontos definícióval, amelyek ismerete a feladat megoldásánál nélkülözhetetlen.

Optimumkritérium alatt azt a célt értjük, amelynek érdekében keressük az optimális megoldást. Az optimumkritérium mátrixa a célmátrix (c_{ij}).

Optimális megoldás alatt azt az **X** mátrixot értjük, amely megmondja, melyik kibocsátóhelyről melyik fogadóhelyre hány rakományegységnyi árut kell elszállítani ahhoz, hogy a szállítás kielégítse az optimumkritériumot.

A program értékén a célfüggvénynek azt az értékét értjük, amelyet az optimális megoldás szolgáltat.

2.2. A célmátrix átalakítása

A konkrét programozási feladat megkezdése előtt esetenként célszerű a célmátrixot átalakítani. Az átalakításnak az az alapgondolata, hogy a megoldás szempontjából indifferens, hogy a célmátrix bármely sorának vagy oszlopának elemeit ugyanazzal a számmal csökkentjük, illetve növeljük. Ezt egy egyszerű példával könnyen bizonyíthatjuk.

Tegyük fel, hogy csak két fogadó- és két kibocsátóhelyről van szó, és a kapacitás, valamint az igény értéke mindenhol egy-egy kocsirakomány. A célmátrix a 2.3. táblázat szerint néz ki (a távolságértékek kilométerben vannak megadva).

2.3. táblázat

	F_1	F_2	
K_1	6	2	1
K_2	4	3	1
	1	1	

Ebben az esetben a lehetséges programok száma kettő. Az első program szerint szállíthatunk az első kibocsátóhelyről az első fogadóhelyre, valamint a második kibocsátóhelyről a második fogadóhelyre. A második program szerint az első kibocsátóhelyről a második fogadóhelyre, a második kibocsátóhelyről az első fogadóhelyre szállítunk. Jelöljük L_1-gyel az első, L_2-vel a második programhoz tartozó távolságot:

$L_1 = 6 + 3 = 9$ km

$L_2 = 4 + 2 = 6$ km

Egyértelmű, hogy a második program jobb, mint az első, mégpedig:

$L_1 – L_2 = 9 – 6 = 3$ kilométerrel.

Most alakítsuk át úgy a kilométermátrixot, hogy a második oszlop minden eleméből vonjunk le 2-t, így az új mátrix a 2.4. táblázat szerinti lesz.

2.4. táblázat

	F_1	F_2	
K_1	6	0	1
K_2	4	1	1
	1	1	

Végezzük el ismételten az előző számítást:

$L_1 = 6 + 1 = 7$

$L_2 = 4 + 0 = 4$

Ismét a második program a jobb, és a szállítási távolságokat összehasonlítva a különbség ismét annyi, mint az előző esetben:

$L_1 - L_2 = 7 - 4 = 3$ km.

Ez a példa már sejteti az alábbi tételt, amelyet matematikailag is bizonyítani fogunk: az optimális program szempontjából bármely szállítási probléma változatlan marad, ha a célmátrix bármely oszlopának vagy sorának elemeit ugyanazzal a számmal növeljük vagy csökkentjük.

Az új mátrix alapján számított célfüggvény persze módosul, mégpedig annyival, amennyivel megváltoznak az oszlopok, illetve sorok értékei.

Bizonyítás: Legyen r_i és s_j tetszőleges konstansok. Képezzük a redukált célmátrix-elemeket az alábbiak szerint:

$$c'_{ij} = c_{ij} - r_i - s_j, \tag{2.8}$$

$$\sum_{i,j} c'_{ij} x_{ij} = \sum_{i,j} \left(c_{ij} + r_i + s_j\right) x_{ij}, \tag{2.9}$$

felbontva a zárójelet:

$$\sum_{i,j} c'_{ij} x_{ij} = \sum_{i,j} c_{ij} x_{ij} + \sum_{i,j} r_i x_{ij} + \sum_{i,j} s_j x_{ij}. \tag{2.10}$$

Mivel r_i és s_j konstansok, ezért kiemelhetőek a szummázás elé:

$$\sum_{i,j} r_i x_{ij} = r_i \sum_j x_{ij}, \qquad \sum_{i,j} s_j x_{ij} = s_j \sum_i x_{ij}. \tag{2.11}$$

Ha a_i-vel jelöljük az i-edik kibocsátóhely kapacitását és b_j-vel a j-edik fogadóhely igényét, akkor igaz az, hogy

$$a_i = \sum_j x_{ij}, \qquad b_j = \sum_i x_{ij}, \tag{2.12}$$

vagyis a (2.10) a következő alakba írható:

$$\sum_{i,j} c'_{ij} x_{ij} = \sum_{i,j} c_{ij} x_{ij} + \sum_{i} r_i a_i + \sum_{j} s_j b_j . \tag{2.13}$$

Tehát $\sum r_i a_i$ és $\sum s_j b_j$ konstansok, így igaz az, hogy a redukált célmátrix alapján számított célfüggvény csak egy konstanssal tér el az eredeti célmátrixból számítottól.

Ezek alapján az optimális megoldás a redukált célfüggvény esetében is ugyanolyan x_{ij} értékek mellett áll fenn, vagyis:

$$\min \sum_{i,j} c'_{ij} x_{ij} = \min \sum_{i,j} c_{ij} x_{ij} + C_R = C_0 + C_R , \tag{2.14}$$

$$C_R = \sum_{i=1}^{m} r_i a_i + \sum_{j=1}^{n} s_j b_j . \tag{2.15}$$

Felhasználva matematikailag is bizonyított tételünket, le lehet egyszerűsíteni célmátrixunkat úgy, hogy minden sorában és oszlopában legyen legalább egy nulla elem. Ezt úgy érjük el, hogy első lépésként megkeressük minden oszlop legkisebb elemét, és ezt kivonjuk az oszlop összes eleméből, majd ugyanezt végrehajtjuk a soroknál is. Ezt a műveletet redukálásnak nevezzük. Előfordulhat az is, hogy már az oszlopok redukálása után találunk minden oszlopban és sorban nulla elemet. Ebben az esetben a soronkénti redukálásnak már nincs értelme.

Nézzük meg példánkon keresztül, hogyan kell a redukálást végrehajtani a 2.1. táblázat mátrixán. Minden oszlopban keressük meg a legkisebb elemet (2, 1, 3, 5, 3), és vonjuk le az oszlop mindegyik eleméből. Ekkor a 2.5. mátrixot kapjuk. Ezt követően soronként is válasszuk ki a legkisebb elemeket (1, 0, 0, 0), és végezzük el itt is a redukálást (2.6. mátrix).

2.5. mátrix

$$\begin{bmatrix} 4 & 1 & 5 & 2 & 2 \\ 2 & 2 & 4 & 0 & 6 \\ 0 & 0 & 0 & 1 & 1 \\ 3 & 5 & 1 & 3 & 0 \end{bmatrix}$$

2.6. mátrix

$$\begin{bmatrix} 3 & 0 & 4 & 1 & 1 \\ 2 & 2 & 4 & 0 & 6 \\ 0 & 0 & 0 & 1 & 1 \\ 3 & 5 & 1 & 3 & 0 \end{bmatrix}$$

Látható, hogy csak az első sorban kellett újból redukálni, így minden oszlopban és sorban van nulla elem. Itt kell megjegyezni, hogy a redukálás végrehajtásánál a sorok és az oszlopok szerepe felcserélhető. Kezdhettük volna az átalakítást a sorokkal is, célszerű azonban ott kezdeni a redukálást, ahol több elem található. A fentieket is figyelembe véve a feladatot leíró új mátrix, kiegészítve a kapacitások és az igények vektoraival, a 2.7. táblázat szerinti lesz.

2.7. táblázat

3	0	4	1	1	40
2	2	4	0	6	70
0	0	0	1	1	60
3	5	1	3	0	30
30	60	50	40	20	200

=

C	**a**
b	$\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$

Ez a mátrix az elszállítandó árumennyiséget tekintve ugyanazt az optimális programot szolgáltatja, mint a 2.1. táblázat mátrixa. A szállítás futásteljesítményét tekintve nem ez a helyzet, a két táblázat alapján számított futásteljesítmény különbségét azonban könnyű kiszámítani. Nem kell mást tennünk, mint az egyes oszlopokból, illetve sorokból levont értékeket megszoroznunk az adott oszlop, illetve sor végén feltüntetett kapacitás vagy igény értékével, majd az így nyert szorzatokat kell összeadnunk. Példánkban ez a következő:

$$C_R = 2 \cdot 30 + 1 \cdot 60 + 3 \cdot 50 + 5 \cdot 40 + 3 \cdot 20 + 1 \cdot 40 + 0 \cdot 70 + 0 \cdot 60 + 0 \cdot 30 = 570 \text{ km}.$$

Ha a redukált célmátrix alapján kiszámított futásteljesítményt az előbb kiszámított összeggel megnöveljük, megkapjuk a végleges, optimális eredményt.

2.3. Megoldás disztribúciós módszerrel

A disztribúciós eljárás lényege, hogy megadunk egy induló programot, majd azt lépésről lépésre addig javítjuk, amíg el nem jutunk az optimális megoldáshoz.

2.3.1. Induló program készítése

Már az induló program elkészítésénél célszerű arra törekedni, hogy a szétosztást a lehető legkisebb futásteljesítmény elérésével valósítsuk meg. Ebből a célból többfajta módszert is kidolgoztak az induló program meghatározására, amelyek közül négyet ismertetünk.

2.3.1.1. Északnyugati sarok módszer

A módszer lényege, hogy mindig arra a c_{ij} elemre programozzuk a lehető legtöbbet, amelyre $i + j$ minimális. Ez azt jelenti, hogy az x_{ij} elemek meghatározásánál az [1,1] elemtől indulunk, ami a mátrix bal felső sarkában található. Ez a hely a térképeken az északnyugati iránynak felel meg, innen a módszer neve.

Nézzük, példánkban hogyan készíthetjük el az induló programot. Az induló elem tehát a c_{11} elem, és programozzunk rá annyit, amennyit csak lehet. Az adott relációban az igény 30 rakományegység, a kibocsátóhelyen pedig 40 rakományegység áll rendelkezésre. Ezért a teljes igényt ki tudjuk innen elégíteni, vagyis a c_{11} elemre 30 egységet programozhatunk. Ezt úgy jelöljük, hogy a megfelelő c_{ij} elemet bekeretezzük, és a megfelelő x_{ij} értéket a c_{ij} jobb fölső sarkába írjuk (2.8. táblázat). Azokat a c_{ij} elemeket, amelyekre $x_{ij} > 0$, kötött elemeknek, amelyekre $x_{ij} = 0$, szabad elemeknek nevezzük.

Mivel az F_1 fogadóhely minden igényét kielégítettük, az első oszlopban nem programozhatunk többet. A fennmaradó c_{ij} mátrixelemek közül a c_{21}-re minimális az $i + j$ összeg, így erre programozunk. A K_1 kibocsátóhelynek még van 10 egységnyi szabad kapacitása, az F_2 fogadóhelynek pedig 60 egység igénye, amelyből tehát 10-et ki tudunk elégíteni: c_{21}-re 10 egységet programozunk: $x_{21} = 10$. Mivel a K_1 kibocsátóhelyen nem maradt több elprogramozható egység, az első sor elemeire a továbbiakban nem programozhatunk.

A fennmaradó elemek közül a c_{22} elemre minimális az $i + j$ összeg, így a következő lépésben erre programozzuk a lehető legtöbbet. Mivel a K_2 kibocsátóhely kapacitása 70, és az F_2 fogadóhely igényéből még 50 egységet nem elégítettünk ki, ezt meg tudjuk tenni: $x_{22} = 50$. Az F_2 fogadóhelyet tehát teljesen kielégítettük, és a K_2 kibocsátóhelyen még maradt 20 elprogramozatlan egység.

A szabályt tovább folytatva, mindig a minimális $i + j$ összegű c_{ij} elemekre a lehető legnagyobb x_{ij} értéket programozva a 2.8. táblázatot kapjuk eredményül.

2.8. táblázat

[6]^{30}	[2]^{10}	8	7	5	~~40~~	~~10~~
4	[3]^{50}	[7]^{20}	5	9	~~70~~	~~20~~
2	1	[3]^{30}	[6]^{30}	4	~~60~~	~~30~~
5	6	4	[8]^{10}	[3]^{20}	~~30~~	~~20~~
~~30~~	~~60~~	~~50~~	~~40~~	~~20~~	200	
	~~50~~	~~30~~	~~10~~			

Sikerült tehát minden igényt kielégítenünk és az összes kapacitást felhasználnunk. Olvassuk most le, hogy az elkészült program alapján végrehajtandó szállítási feladatnak mekkora a futásteljesítménye. Mint már említettük, az egyes relációkban jelentkező futásteljesítményeket úgy tudjuk meghatározni, hogy az egyes kötött elemeket (kilométerelemek, c_{ij}) megszorozzuk az adott viszonylatban elszállított rakományegységekkel (x_{ij}), majd a szorzatok eredményét az egész feladatra vonatkozóan összeadjuk.

A 2.8. táblázat alapján ez a következőképpen történik:

$K_1 – F_1$ relációban: $30 \cdot 6 = 180$ km,
$K_1 – F_2$ relációban: $10 \cdot 2 = 20$ km,
$K_2 – F_2$ relációban: $50 \cdot 3 = 150$ km,
$K_2 – F_3$ relációban: $20 \cdot 7 = 140$ km,
$K_3 – F_3$ relációban: $30 \cdot 3 = 90$ km,
$K_3 – F_4$ relációban: $30 \cdot 6 = 180$ km,
$K_4 – F_4$ relációban: $10 \cdot 8 = 80$ km,
$K_4 – F_5$ relációban: $20 \cdot 3 = 60$ km.

Tehát $C = 900$ km.

Mivel az **X** mátrix többi eleme 0, ezért a Schur-szorzatba nem adnak tagot.

Ha a program értékét a redukált mátrix felhasználásával határoztuk volna meg, C értékére egy kisebb értéket kaptunk volna, amelyet a redukáláskor kiszámított C_R értékkel még meg kellett volna növelni.

2.3.1.2. Progresszív módszer

Az úgynevezett progresszív módszer a költségmátrix aktuálisan legkisebb elemeit felhasználva szolgáltat induló programot: minden lépésben megkeressük a c_{ij} mátrix legkisebb elemét, és a lehető legnagyobb értéket programozzuk rá. Két azonos c_{ij} mátrixelem esetén azt választjuk, amelyikre többet lehet programozni. Ha ez is azonos, tetszőlegesen választunk közülük.

A 2.1. mátrix legkisebb eleme a c_{32} elem, értéke 1. Erre programozzuk tehát elsőként a lehető legnagyobb értéket, 60-at: az F_2 fogadóhely igénye 60 egység, amelyet a K_3 kibocsátóhely teljes egészében ki tud elégíteni. A következő legkisebb értékű eleme a költségmátrixnak a 2, amely a c_{12} és a c_{31} helyeken fordul elő. Azonban sem a c_{12}, sem a c_{31} elemre nem lehet már programozni. A következő legkisebb mátrixelemek a $3 = c_{22} = c_{33} = c_{45}$, amelyek közül csak a c_{45}-re lehet programozni, 20 egységet. Mind a c_{21}, mind a c_{43} elem értéke 4, amely a következő legkisebb költségmátrixelem, amelyekre még lehet programozni. Azonban míg a c_{43} elemre csak 10 egységet, addig a c_{21} elemre 30 egységet lehet programozni, ezért ezt választjuk. A következő lépésben már programozhatunk a c_{43} elemre, hiszen ez a legkisebb a lehetséges elemek közül. Elvégezve a programozást a c_{24} elemre 40-et, a c_{13} elemre pedig szintén 40 egységet programozva, a 2.9. táblázatot kapjuk, amelyből leolvasható a program értéke.

2.9. táblázat

6	2	[8]^{40}	7	5	~~40~~	
[4]^{30}	3	7	[5]^{40}	9	~~70~~	~~40~~
2	[1]^{60}	3	6	4	~~60~~	
5	6	[4]^{10}	8	[3]^{20}	~~30~~	~~10~~
~~30~~	~~60~~	~~50~~ ~~40~~	~~40~~	~~20~~	200	

$C = 8 \cdot 40 + 4 \cdot 30 + 5 \cdot 40 + 1 \cdot 60 + 4 \cdot 10 + 3 \cdot 20 = 800$ km

2.3.1.3. Kettős előny módszer

Az úgynevezett kettős előny módszer a progresszív módszerhez képest annyiban más, hogy oszloponként és soronként veszi figyelembe a legkisebb elemet.

Első lépésben aláhúzzuk minden sor legkisebb elemét. Ezután, függetlenül attól, hogy melyik elem van már aláhúzva, aláhúzzuk minden oszlop legkisebb elemét is. Ezzel lesznek kétszer, egyszer aláhúzott és aláhúzatlan elemeink. A kétszer aláhúzott elem(ek) az adott oszlopban és sorban is a legkisebb elem(ek), az egyszer aláhúzott(ak) pedig vagy csak a soruknak, vagy csak az oszlopuknak a legkisebb eleme(i). Ezt láthatjuk a 2.10. táblázatban.

2.10. táblázat

6	2	8	7	5	40
4	3	7	5	9	70
2	1	3	6	4	60
5	6	4	8	3	30
30	60	50	40	20	200

Első körben a kétszer aláhúzott elemekre programozunk, innen a módszer neve. Ezen elemek közül kiválasztjuk a legkisebbet, ha több ilyen elem van, akkor arra programozunk, amelyikre nagyobb értéket lehet. Ha ez is azonos, akkor szabadon választhatunk. Most két darab ilyen kétszer aláhúzott elemünk van: $c_{31} = 1$ és $c_{45} = 3$. A c_{31} elem a kisebb, tehát erre programozzuk a lehető legtöbbet, 60-at. Ezután a c_{45} elemre még lehet programozni, ezért 20 egységet ráprogramozunk.

Ezzel elfogytak a kétszer aláhúzott elemek, ezért második körben áttérünk az egyszer aláhúzott elemekre. A választás további elvei ugyanazok, mint eddig. A legkisebb egyszer aláhúzott elemek a $c_{12} = c_{31} = 2$, de az egyiknek a sorában, a másiknak az oszlopában nincs szabad elprogramozható kapacitás, illetve igény. Ezért továbblépünk a következő legkisebb egyszer aláhúzott elemre, a 3 értékű c_{22}-re és c_{33}-ra, azonban ezekkel is hasonló a helyzet, nincs szabad kapacitásuk, illetve igényük. A maradék egy egyszer aláhúzott elemnek, a c_{24}-nek a sorában azonban van 40 egység kapacitás, oszlopában 40 egység igény, így erre rá tudunk programozni 40 egységet.

Ezzel elfogytak az egyszer aláhúzott elemek is. Utolsó körben az aláhúzatlan elemekre programozunk: a c_{21}-re 30-at, a c_{43}-ra 10-et, a c_{13}-ra pedig 40-et. Ezzel minden kapacitást elprogramoztunk, minden igényt kielégítettünk. A program értéke a 2.11. táblázatból leolvasható.

2.11. táblázat

6	2	8 40	7	5	~~40~~	
4 30	3	7	5 40	9	~~70~~	~~40~~
2	1 60	3	6	4	~~60~~	
5	6	4 10	8	3 20	~~30~~	~~10~~
~~30~~	~~60~~	~~50~~	~~40~~	~~20~~	200	
		~~40~~				

$C = 8 \cdot 40 + 4 \cdot 30 + 5 \cdot 40 + 1 \cdot 60 + 4 \cdot 10 + 3 \cdot 20 = 800$ km

2.3.1.4. Bástyamódszer

Ezen módszer alkalmazásakor két szabályt kell betartanunk:

1. A mátrixon úgy haladjunk végig, hogy miután egy c_{ij} elemre programoztunk, a következő elem, amelyre programozunk, az *i*-edik sorban vagy a *j*-edik oszlopban legyen. Ezzel az egymás után programozott elemek közti mozgás úgy néz ki, mint ahogy a bástya mozog a sakktáblán, innen a módszer neve.
2. Mindig a legkisebb szabad elemre programozunk, és mindig a lehető legnagyobb mennyiséget.

Itt mindjárt felmerül az a probléma, hogy a minimális költségelem nem mindig választható ki egyértelműen, a programozás kezdetén pedig az, hogy melyik elemre programozzunk elsőként. A 2.7. táblázat redukált mátrixára programozva a minimális kilométerelem 0, amelyből minden sorban és oszlopban megtalálható legalább egy. Célszerű azt a nullát választanunk, amelynek sorában vagy oszlopában több nulla már nincs. Ha ilyen nulla elem nem található, akkor olyan nullát kell választani, amely mellett a lehető legkevesebb nulla elem van. Nézzük, példánkban hogyan készíthetjük el az induló programot. Válasszuk induló elemnek a c_{24} elemet, és programozzunk rá annyit, amennyit csak lehet. Az adott relációban az igény 40 rakományegység, a kibocsátóhelyen pedig 70 rakományegység áll rendelkezésre. Ezért a teljes igényt ki tudjuk elégíteni, vagyis a c_{24} elemre 40 egységet programozhatunk (2.12. táblázat).

Mivel a F_4 fogadóhely minden igényét kielégítettük, a bástyaszabályt figyelembe véve az oszlopban továbbhaladni nem tudunk, így az induló

irányra merőlegesen haladunk tovább a második sorban. Itt minimális elem mind a c_{21}, mind a c_{22}, éppen ezért bármelyiket választhatnánk. Nem célszerű azonban a c_{21} elem választása, mert ebben az esetben degeneráció lépne fel (a degeneráció problémájával a következő alpontban részletesen foglalkozunk). Programozzuk tehát a c_{22} elemre. Az F_2 fogadóhely igénye jóllehet 60 egység, azonban csak a K_2 kibocsátóhely maradékát, vagyis 30 egységet tudunk erre az elemre programozni.

2.12. táblázat

3		[0]^{30}	→	4	→	1	→	[1]^{10}		40
		↑						↓		
2		[2]^{30}	←	4	←	[0]^{40}		6		70
								↓		
[0]^{30}	←	0	←	[0]^{30}		1		1		60
				↑				↓		
3		5		[1]^{20}	←	3	←	[0]^{10}		30
30		60		50		40		20		200

Ezt a szisztémát folytatva a 2.12. táblázatban megjelölt sorrendben tudjuk elvégezni a programozást. Sikerült minden igényt kielégítenünk és az összes kapacitást felhasználnunk. Olvassuk most le, hogy az elkészült program alapján végrehajtandó szállítási feladatnak mekkora a futásteljesítménye. A 2.12. táblázat alapján ez a következőképpen történik:

$K_1 – F_2$ relációban: $30 \cdot 0 = 0$ km
$K_1 – F_5$ relációban: $10 \cdot 1 = 10$ km
$K_2 – F_2$ relációban: $30 \cdot 2 = 60$ km
$K_2 – F_4$ relációban: $40 \cdot 0 = 0$ km
$K_3 – F_1$ relációban: $30 \cdot 0 = 0$ km
$K_3 – F_3$ relációban: $30 \cdot 0 = 0$ km
$K_4 – F_3$ relációban: $20 \cdot 1 = 20$ km
$K_4 – F_5$ relációban: $10 \cdot 0 = 0$ km
Összesen: 90 km

Tehát $C_0 = 90$ km.

Ez természetesen nem a valódi értéket mutatja, mivel a redukált mátrixból számítottuk ki. C valós értékét úgy kaphatjuk meg, hogy a redukáláskor kiszámított C_R értékkel még megnöveljük: $C = C_R + C_0 = 570 + 90 = 660$ km.

Alternatív lehetőség a program valós értékének kiszámítására, hogy a programozott x_{ij} értékeket nem a redukált mátrix elemeivel, hanem az eredeti c_{ij} kilométerelemekkel szorozzuk meg. A bemutatott példában mindkét módszerrel határozzuk meg a célfüggvény értékét. A 2.1. táblázatban szereplő eredeti kilométerelemeket felhasználva az egyes szállítási viszonylatok, majd a teljes feladat futásteljesítményét az alábbiak szerint határozzuk meg.

$K_1 – F_2$ relációban: 30·2 = 60 km
$K_1 – F_5$ relációban: 10·5 = 50 km
$K_2 – F_2$ relációban: 30·3 = 90 km
$K_2 – F_4$ relációban: 40·5 = 200 km
$K_3 – F_1$ relációban: 30·2 = 60 km
$K_3 – F_3$ relációban: 30·3 = 90 km
$K_4 – F_3$ relációban: 20·4 = 80 km
$K_4 – F_5$ relációban: 10·3 = 30 km
Összesen: 660 km

Tehát C = 660 km.

2.3.2. Degeneráció és megszüntetése

Bizonyítás nélkül mondjuk ki, hogy a kötött elemek száma minden esetben 1-gyel kisebb a mátrix sorainak és oszlopainak összegénél. Ezt a számot nevezzük kritikus számnak:

$$k = m + n - 1, \tag{2.16}$$

ahol k a kritikus szám, m a célmátrix sorainak a száma, n a célmátrix oszlopainak a száma.

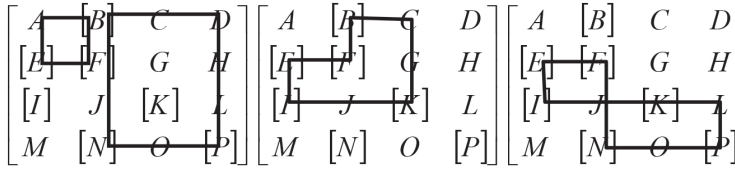
Példánkban a mátrix sorainak és oszlopainak száma összesen kilenc, vagyis nyolc kötött elemnek kell lennie. Ez az északnyugati sarok módszernél teljesült is, a progresszív és a kettős előny módszer esetében azonban csak hat-hat kötött elemet kaptunk, azaz ezek a megoldások kétszeresen degeneráltak, mivel a kötött elemek száma kettővel kisebb a kritikus számnál. A program optimalitásának ellenőrzéséhez és javításához ezt a degenerációt meg kell szüntetni.

Ez mindössze annyit jelent, hogy két szabad elemet kötött elemmé teszünk úgy, hogy 0 értéket programozunk rájuk. Hogyan válasszuk ki, melyik legyen ez a két elem? A szabály, hogy úgynevezett független elemeket

válasszunk. Független elemnek nevezzük azokat a nem kötött elemeket, amelyekből kiindulva nem tudunk zárt hurkot rajzolni.

A hurok rajzolásának menete a következő. Elindulunk egy szabad elemről, és útközben csak kötött elemeken mozogva felváltva vízszintes és függőleges irányban megpróbálunk visszatérni a kiinduló szabad elemre. Egy elmozdulás során nem kötelező az adott irányban legközelebbi kötött elemre lépni, át is lehet kötött elemeket ugrani, ekkor azok nem részei a huroknak. Lehetséges továbbá, hogy a hurok metssze önmagát, azaz egy nyolcasszerű módon fusson az elemeken, ez is teljesen szabályos forma. A 2.1. ábrán láthatunk néhány lehetséges hurkot egy hipotetikus mátrixban az *A*, *D*, *C*, *L* elemekre.

2.1. ábra



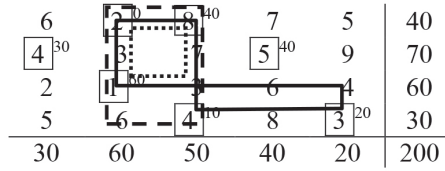
A 2.13. táblázat azonos a progresszív módszerrel kapott induló program mátrixával, a 2.9. táblázattal. Ebben csak egy nem független elem van, a c_{15} elem, mivel ez az egyetlen, amelyből kiindulva tudunk zárt hurkot rajzolni. Ezen kívül bármelyiket választhatjuk az első kötött elemnek, amelyre nullát programozunk.

2.13. táblázat

6	2	8 40	7	5	40
4 30	3	7	5 40	9	70
2	1 60	3	6	4	60
5	6	4 10	8	3 20	30
30	60	50	40	20	200

Válasszuk a tizenhárom lehetséges elem közül a c_{12}-t, és programozzunk rá nullát (2.14. táblázat). Ezzel a c_{15} mellett három új nem független elemünk lett: c_{33}, c_{35} és c_{42}. A c_{35} hurkán (folytonos vonal) láthatjuk, hogy önmagát keresztezi a hurok, a c_{42}-én pedig (szaggatott vonal), hogy „átlép” egy kötött elemet, a c_{32}-t.

2.14. táblázat



6	2 0	8 40	7	5	40
4 30	3	7	5 40	9	70
2	1 60	3	6	4	60
5	6	4 10	8	3 20	30
30	60	50	40	20	200

Maradt tehát kilenc független elem, amelyet nulla értékkel bevonhatunk a programba a degeneráció megszüntetése érdekében. Válasszuk például a c_{31} elemet. Ekkor a programot leíró mátrix a 2.15. táblázat szerinti lesz, amelyben már nincs független elem.

2.15. táblázat

6	2 0	8 40	7	5	40
4 30	3	7	5 40	9	70
2 0	1 60	3	6	4	60
5	6	4 10	8	3 20	30
30	60	50	40	20	200

A kötött elemek száma most már valóban eggyel kevesebb a sorok és oszlopok összdarabszámánál, így a program optimalitása ellenőrizhető, és ha szükséges, javítható.

2.3.3. A program javítása

Az induló program elkészítése általában nem ad optimális eredményt, bár gyakran igen jó megoldást szolgáltat. Ahhoz, hogy a végeredményt megkapjuk, javítani kell a programot.

Abban az esetben, ha a programban minden redukált célmátrixbeli kötött elem értéke nulla lenne, akkor máris optimális megoldással rendelkeznénk, hiszen nem volna lehetőségünk az összes futásteljesítmény további csökkentésére. Ez azonban ritkán fordul elő, így meg kell vizsgálnunk, van-e mód a program javítására.

A javítást több módszerrel is el lehet végezni. Itt most két módszert fogunk részletesen bemutatni, a potenciálok módszerét *(MODI method)*

és a hurokmódszert *(stepping stone method).* Ezek az eljárások bonyolultabb feladatok esetében is viszonylag egyszerűek és jól áttekinthetők.

2.3.3.1. Potenciálok módszere

Mielőtt a javítás részletes menetére rátérnénk, ismerkedjünk meg a potenciál fogalmával. Potenciáloknak nevezzük a továbbiakban azokat a mátrix egy sorához, illetve oszlopához rendelt számokat, amelyeknek tulajdonsága, hogy minden kötött elemre a potenciálok összege egyenlő a kötött elem értékével.

A sorokhoz rendelt potenciálokat a továbbiakban u_1, u_2,…, u_m, az oszlopokhoz rendelteket v_1, $v_{2,...}$, v_n-nel jelöljük, azaz a fenti definíció értelmében minden célmátrixbeli kötött elemre igaz a $c_{ij} = u_i + v_j$ összefüggés. Egy potenciál értékét szabadon választhatjuk meg, a többit azonban számítanunk kell, felhasználva a választott potenciált. Célszerű az első potenciált nullának venni, és abban a sorban vagy oszlopban választanunk, ahol a legtöbb kötött elem található.

Válasszuk az első oszloppotenciált nullának. Ekkor a második sorpotenciálnak 4-nek, a harmadik oszloppotenciálnak pedig 2-nek kell lennie, hogy összegük a megfelelő kötött elem értékét ($c_{21} = 0 + 4$, $c_{32} = 0 + 2$) kiadja. A második sorpotenciálból kiindulva kapjuk a negyedik oszloppotenciál értékét: $4 + v_4 = 5$, ahonnan a $v_4 = 1$. Hasonlóan meghatározható a harmadik sorpotenciál segítségével a második oszloppotenciál (–1), abból az első sorpotenciál (3), majd a harmadik oszloppotenciál (5), ebből a negyedik sorpotenciál (–1) és végül az ötödik oszloppotenciál (4). A potenciálokat a 2.16. táblázatban láthatjuk.

2.16. táblázat

u_i \ v_j	0	–1	5	1	4
3	6	[2]	[8]	7	5
4	[4]	3	7	[5]	9
2	[2]	[1]	3	6	4
–1	5	6	[4]	8	[3]

Meg kell jegyezni, hogy a sorpotenciálokhoz bármilyen *A* számot hozzáadva, az oszloppotenciálokból pedig ugyanezt az *A* értéket levonva a megoldás további menete nem változik meg.

A következő lépésben képezzük az úgynevezett differenciákat minden mátrixelemre. A differenciák mátrixát K_{ij}-vel jelöljük. Ezek segítségével dönthetjük el, hogy mely szabad elemeket érdemes bevonni a programba. A differenciák értéke az adott mátrixelem mínusz a hozzá tartozó oszlop- és sorpotenciál: $K_{ij} = c_{ij} - u_i - v_j$. A kötött elemek differenciája tehát a potenciálok definíciója szerint mindig 0 lesz. Az összes differencia értékét a 2.17. mátrixban láthatjuk.

2.17. mátrix

$$K_{ij} = \begin{bmatrix} 3 & \boxed{0} & \boxed{0} & 3 & -2 \\ \boxed{0} & 0 & -2 & \boxed{0} & 1 \\ \boxed{0} & \boxed{0} & -4 & 3 & -2 \\ 6 & 8 & \boxed{0} & 8 & \boxed{0} \end{bmatrix}$$

A differencia értéke a következő információkat tartalmazza:

ha K_{ij} pozitív, akkor az adott c_{ij} elem bevonása nem javítaná a programot, hanem rontaná,

ha K_{ij} negatív, akkor az adott elem bevonása javítja a programot,

ha $K_{ij} = 0$, az adott c_{ij} elem bevonása alternatív megoldást ad.

Mint már említettük, a differenciákat minden szabad elemre meghatározzuk. Azt az elemet célszerű bevonni a programba, amelyikre a legnegatívabb értéket kapjuk. A differenciák meghatározása azonban csak arra szolgált, hogy megállapítsuk, mely elemeket lehet bevonni a program javításába. Látható, hogy a differenciák között van több negatív szám is, a legkisebb ezek közül a $K_{33} = -4$, így a c_{33} elem bevonása javítja legnagyobb mértékben a programot.

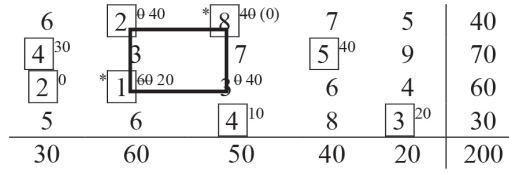
A javításhoz ismét hurkot rajzolunk, most a bevonni kívánt elemből kiindulva. Ez jelen esetben egyszerű, ahogy a 2.18. táblázat is mutatja.

2.18. táblázat

6	2 [0]	8 [40]	7	5	40
4 [30]	3	7	5 [40]	9	70
2 [0]	1 [60]	3	6	4	60
5	6	4 [10]	8	3 [20]	30
30	60	50	40	20	200

A hurok megrajzolása után a szabad elemből kiindulva a következő sarokpontot egy csillaggal megjelöljük, majd minden második elemet is. Ezután a csillagos elemek közül kiválasztjuk azt, amelyikre a legkevesebbet programoztuk (2.19. táblázat).

2.19. táblázat



Példánkban ez a c_{13} elem, amire $x_{13} = 40$-et programoztunk. Ezt az értéket vonjuk le a csillagos elemekből, és adjuk hozzá a meg nem csillagozott elemekhez. Így a c_{13} elem szabad elemmé válik, míg az eredetileg szabad c_{23} elem kötött elem lesz. Az így átalakított új táblázat a 2.20. táblázat.

2.20. táblázat

6	2 0	8	7	5	40
4 30	3	7	5 40	9	70
2 0	1 60	3 40	6	4	60
5	6	4 10	8	3 20	30
30	60	50	40	20	200

A javított program célfüggvényének értéke a következő:

$$C = 2 \cdot 40 + 4 \cdot 30 + 5 \cdot 40 + 2 \cdot 0 + 1 \cdot 20 + 3 \cdot 40 + 4 \cdot 10 + 3 \cdot 20 = 640 \text{ km}$$

Összehasonlítva ezt az eredményt az induló programnál kapott eredménnyel, láthatjuk, hogy ez valóban kedvezőbb.

A programozási feladatnak azonban még nincs vége, hiszen nem biztos, hogy az optimális eredményt kaptuk meg. Vizsgáljuk meg, hogy a kapott eredmény tovább javítható-e. A 2.20. táblázat felhasználásával ismételten képezzünk potenciálokat, majd határozzuk meg az egyes szabad elemekhez tartozó differenciákat. Ezt szemlélteti a 2.21. táblázat.

2.21. táblázat

u_i \ v_j	0	–1	1	1	0
3	3	[0]	4	3	2
4	[0]	0	2	[0]	5
2	[0]	[0]	[0]	3	2
3	2	4	[0]	4	[0]

$= K_{ij}$

Látható, hogy a differenciák mátrixa nem tartalmaz negatív értéket, így az optimális programot kaptuk meg. Ha a differenciák között található nem kötött elemhez tartozó 0 érték (mint esetünkben a K_{22}), akkor a programnak alternatív megoldása van. Ez azt jelenti, hogy az adott elem bevonásával csak az egyes viszonylatokban szállított árumennyiség értéke változik, de magának a programnak az értéke nem.

2.3.3.2. Hurokmódszer

Ennél a módszernél csak hurkokat keresünk: minden nem kötött elemre meghatározzuk a hozzá tartozó zárt hurkot, majd ennek elemeit váltakozó előjellel összeadjuk.

Vegyük ismét alapul a 2.9. táblázatot, és rajzoljunk egy hurkot az első szabad elemre, c_{11}-re: $c_{11} - c_{12} - c_{32} - c_{31} - c_{11}$. A hurok kiinduló eleméből, a szabad elemből a hurok első elemét kivonva, a másodikat hozzáadva, a harmadikat ismét kivonva stb., megkapjuk a hurok értékét:

$K_{11} = 6 - 2 + 1 - 2 = 3.$

A számolást elvégezve az összes lehetséges hurokra:

$K_{14} = 7 - 5 + 4 - 2 + 1 - 2 = 3$
$K_{15} = 5 - 3 + 4 - 8 = -2$
$K_{22} = 3 - 1 + 2 - 4 = 0$
$K_{23} = 7 - 8 + 2 - 1 + 2 - 4 = -2$
$K_{25} = 9 - 3 + 4 - 8 + 2 - 1 + 2 - 4 = 1$
$K_{33} = 3 - 8 + 2 - 1 = -4$
$K_{34} = 6 - 5 + 4 - 2 = 3$
$K_{35} = 4 - 3 + 4 - 8 + 2 - 1 = -2$
$K_{41} = 5 - 2 + 1 - 2 + 8 - 4 = 6$
$K_{42} = 6 - 2 + 8 - 4 = 8$
$K_{44} = 8 - 5 + 4 - 2 + 1 - 2 + 8 - 4 = 8$

Mint azt a 2.17. mátrixszal összehasonlítva láthatjuk, nem véletlenül jelöltük az egyes hurkok értékeit a differenciákkal azonosan *K*-val, mivel ezzel a módszerrel is ugyanazokat az értékeket kaptuk.

Éppen ezért a folytatás is azonos: a legnegatívabb hurok kezdőpontját vonjuk be a programba. Ennek a huroknak a bevonandó elemtől mint első elemtől számított minden páros elemét megcsillagozzuk, majd kiválasztjuk a legkisebb, csillagos elemre programozott x_{ij} értéket. Ezt az x_{ij} értéket a csillagos elemek x értékéből levonjuk, a csillagtalanokéhoz hozzáadjuk. Ezzel az a csillagos elem, amelyikre a legkevesebb volt eddig programozva, szabad elemmé válik, azaz a kötött elemek száma a javítás során állandó.

Ezután a hurkok értékének ismételt kiszámításával ellenőrizzük a javított program optimalitását, és amennyiben ismét találunk negatív értékű hurkot, annak kezdőelemével újra javítjuk a programot. Ezt addig ismételjük, amíg csak pozitív vagy nulla értékű hurkok maradnak. A nulla értékű hurok kiinduló eleme alternatív megoldást jelent: azt az elemet bevonva a program értéke nem, csak az egyes kibocsátóhely-fogadóhely relációk változnak meg.

2.3.4. Összetett szállítási feladatok

A fejezet elején említettük, hogy az egyszerű szállítási probléma nem alkalmas a gyakorlatban jelentkező feladatok megoldására. Ezért ebben a részben megismerkedünk a szállítási probléma speciális eseteivel és azok megoldásának módszereivel.

2.3.4.1. Névleges fogadó- vagy kibocsátóhely beiktatása

Az általános szállítási probléma tárgyalásánál feltételeztük, hogy a fogadóhelyek igénye megegyezik a kibocsátóhelyeken rendelkezésre álló kapacitással. Ez a gyakorlatban igen ritkán fordul elő. Jellemzőbb az az eset, hogy a feladóhelyen nagyobb a kapacitás, vagy a fogadóhelyen nagyobb az igény. Ilyen esetekben a különbségnek megfelelő árumennyiséggel egy új feladóhelyet, illetve fogadóhelyet iktatunk be a programba. Az ilyen helyet nevezzük névleges fogadó- vagy névleges kibocsátóhelynek.

Foglalkozzunk először a névleges fogadóhellyel. Névleges fogadóhelyet akkor kell beállítanunk a programba, ha a kibocsátóhelyek kapacitása meghaladja a fogadóhelyek igényét. Módosítsuk a 2.1. táblázat adatait oly

módon, hogy az F_4 fogadóhely igényét 40 kocsirakományról 25-re csökkentjük. Mivel a feladat matematikai modellje egyenlőséget követel, ezért be kell állítanunk egy hatodik fogadóhelyet, az úgynevezett névleges fogadóhelyet (F_n), 15 rakományegység szükséglettel (2.22. táblázat).

2.22. táblázat

	F_1	F_2	F_3	F_4	F_5	$\mathbf{F_n}$	
K_1	6	2	8	7	5	0	40
K_2	4	3	7	5	9	0	70
K_3	2	1	3	6	4	0	60
K_4	5	6	4	8	3	0	30
	30	60	50	**25**	20	**15**	200

Látható, hogy a névleges fogadóhely oszlopában a kilométerelemek, azaz a c_{ij} értékek helyére mindenhol nullát írtunk. Ezt azért tehetjük meg, mert a névleges fogadóhely szempontjából közömbös, hogy melyik kibocsátóhelyről „elégítik ki" az igényét. Ez azt is implikálja, hogy a feladat megoldásánál a nullára redukálást csak oszloponként kell végrehajtani, hiszen a névleges fogadóhely beiktatásával minden sorban lett nulla elem.

Névleges kibocsátóhelyet akkor kell beiktatnunk a programba, ha a kibocsátóhelyek árukészlete kevesebb, mint a fogadóhelyek áruszükséglete. Módosítsuk most a 2.1. táblázatot úgy, hogy az F_3 fogadóhely igényét növeljük meg 80 rakományegységre. Ez az összkapacitást figyelembe véve (200 rakományegység) 30 rakományegység többletigényt jelent, amit nem tudunk kielégíteni. Hogy a feladat mégis megoldható legyen, névleges kibocsátóhelyet (K_n) iktatunk be a programba, 30 rakományegység kapacitással. A névleges kibocsátóhely sorában, a már említett okok miatt, a kilométerelemek értékei nullák lesznek. A mátrix redukálását ezért most csak soronként kell elvégezni (2.23. táblázat).

2.23. táblázat

	F_1	F_2	F_3	F_4	F_5	
K_1	6	2	8	7	5	40
K_2	4	3	7	5	9	70
K_3	2	1	3	6	4	60
K_4	5	6	4	8	3	30
$\mathbf{K_n}$	0	0	0	0	0	**30**
	30	60	**80**	40	20	200

A feladat megoldása a mátrix névleges állománnyal történő kiegészítése, valamint az esetleges nullára való redukálás után az egyszerű szállítási probléma megoldásánál megismert módszerrel történik.

2.3.4.2. Szállítást korlátozó feltételek

Reláció kizárása

Gyakorlati feladatok megoldása során előfordulhat, hogy egy bizonyos kibocsátóhelyről – valamilyen ok miatt – egy adott fogadóhelyre egyáltalán nem lehet szállítani.

Tételezzük fel, hogy a már említett példánkban a K_2 kibocsátóhelyről F_3 fogadóhelyre nem lehet szállítani (például útlezárás miatt). Ez azt jelenti, hogy a programozás során az x_{23} értéke mindenképpen nulla kell hogy legyen. Ennek érdekében a $K_2 - F_3$ relációban a c_{23} kilométerelem helyére egy végtelen nagy számot szimbolizáló „M" elemet írunk (2.24. táblázat).

2.24. táblázat

	F_1	F_2	F_3	F_4	F_5	
K_1	6	2	8	7	5	40
K_2	4	3	**M**	5	9	70
K_3	2	1	3	6	4	60
K_4	5	6	4	8	3	30
	30	60	50	40	20	200

Nyilvánvaló, hogy a programozás során ez a reláció feltétlenül kiesik, hiszen minden más eleme a mátrixnak ennél az M „számnál" kisebb.

A programozási feladatot az adott reláció kizárása után a már ismertetett módon kell végrehajtani. Amennyiben több kilométerelem helyébe kell beírni az M szimbólumot, előfordulhat, hogy valamelyik M értékre mégis kellene programozni. Ez azt jelenti, hogy a korlátozás nem valósítható meg, vagyis a feladatnak így nincs megoldása.

Kapacitáskorlátok

Igen gyakran előfordulhat a gyakorlatban, hogy egy adott viszonylatban megszabják, korlátozzák a szállítandó áru mennyiségét.

A már eddig is vizsgált példánkban szabjuk meg feltételként, hogy a $K_2 - F_3$ relációban minimum 30 rakományegységet szállítani kell, mert ez a két fél külön szerződésben az adott mennyiség szállítására már előzőleg megállapodott. Annak érdekében, hogy a feladatot meg tudjuk oldani, az induló táblázatot módosítani kell a következő módon: mind a K_2 kibocsátóhely kapacitását, mind az F_3 fogadóhely igényét 30 rakományegységgel csökkenteni kell (2.25. táblázat).

2.25. táblázat

	F_1	F_2	F_3	F_4	F_5	
K_1	6	2	8	7	5	40
K_2	4	3	7	5	9	**40**
K_3	2	1	3	6	4	60
K_4	5	6	4	8	3	30
	30	60	**20**	40	20	200

A feladat megoldásának további menete az egyszerű szállítási feladatnál már ismertetett módon történik. Ha a c_{23} elemre ennek során programozunk, azaz $x_{23} > 0$ lesz, az csak annyit jelent, hogy nem pontosan a minimális mennyiséget szállítottuk ebben a viszonylatban, hanem többet, ami természetesen nem gond, mivel ezt a lehetőséget nem zártuk ki. Fontos azonban megjegyezni, hogy a célfüggvény felírását minden esetben a levont mennyiség és a hozzá tartozó c_{ij} elem szorzatával kell kezdeni. Ha ezt elmulasztjuk, akkor a célfüggvény nem az optimális megoldást mutatja:

$$C = 30 \cdot 7 + \sum x_{ij}\, c_{ij}$$

Nézzünk most egy újabb korlátozó feltételt. Meghatározhatjuk azt is, hogy egy adott viszonylatban pontosan mennyi árut kell szállítani. Példánkban kössük ki azt, hogy a $K_2 - F_3$ relációban 30 rakományegységet kell szállítani, se többet, se kevesebbet. A feladat megoldhatósága érdekében az előző példához hasonlóan kell módosítani az induló mátrixunkat. Itt is le kell vonni az adott relációban mind a kapacitásból, mind az igényből a 30 egységet, azt, amit az adott viszonylatban biztosan el fogunk szállítani. Ezzel

a módosítással azonban még nem tudunk eleget tenni a korlátozó feltételnek. Ki kell zárni a további programozásból a $K_2 - F_3$ viszonylatot, hiszen erre az elemre többet már nem programozhatunk. Hogy a feltétel teljesüljön, a c_{13} elem helyére egy kizáró M-et írunk (2.26. táblázat). A feladat megoldásának további menete a már ismertetett módon történik.

2.26. táblázat

	F_1	F_2	F_3	F_4	F_5	
K_1	6	2	8	7	5	40
K_2	4	3	**M**	5	9	**40**
K_3	2	1	3	6	4	60
K_4	5	6	4	8	3	30
	30	60	**20**	40	20	200

Támasszuk most azt a feltételt a megadott feladattal szemben, hogy $K_2 - F_3$ relációban legfeljebb 30 rakományegységet lehet szállítani, vagyis az $x_{23} \leq 30$ feltétel teljesülését követeljük meg. Ezt a problémát már nem oldhatjuk meg pusztán a kilométerelem és/vagy a kapacitások megváltoztatásával. A feladat megoldásánál a 2.1. táblázatot változatlanul vesszük alapul. Az induló programot is az egyszerű szállítási problémáról tanult módon készítjük el, azzal a módosítással, hogy a programozást a c_{23} elemmel, tehát a korlátozásnak alávetett elemmel kezdjük. Erre programozzuk az adott felső korlátot, ami programunkban: $x_{23} = 30$ (2.27. táblázat). A programozás további menete a már megismert módon történik.

2.27. táblázat

	F_1	F_2	F_3	F_4	F_5	
K_1	6	2	8	7	5	40
K_2	4	3	~~7~~ 30	5	9	70
K_3	2	1	3	6	4	60
K_4	5	6	4	8	3	30
	30	60	50	40	20	200

Mielőtt azonban továbblépnénk a programozásban, ismerkedjünk meg a báziselem fogalmával. Báziselemnek nevezzük azokat a kötött elemeket, amelyekre történő programozáskor egy sor vagy oszlop kiesik. Példánkban látható, hogy a c_{23} elem nem ilyen, ezért ennél az elemnél a többi kötött elemétől eltérő jelölést alkalmaztunk, ami arra utal, hogy a c_{23} kötött elem,

de a többi kötött elemmel szemben nem báziselem. A program javításánál ez azzal jár együtt, hogy a potenciálok meghatározásánál, illetve a hur(k)ok megrajzolásánál nem tekintjük kötött elemnek (nem vonatkoznak rá a kötött elemre vonatkozó szabályok). Vagyis programunkban a c_{23} elemen kívül még nyolc kötött elemnek kell lennie.

A fentiek alapján logikus, hogy az optimumfeltételt meg kell vizsgálni a nem báziselem kötött elemekre is, ami azt jelenti, hogy a c_{23} elemre is meghatározzuk a differencia értékét. Egy nem bázis kötött elem akkor vonható be a programba, ha a hozzá tartozó differencia értéke pozitív. Amennyiben ilyen elemet vonunk be a programba, a hurok csillagozása ezen az elemen kezdődik, annak érdekében, hogy itt a programozott értékből le kelljen vonni, hiszen hozzáadni már nem lehet. A javítás további menete a már ismertetett módon történik.

2.3.5. Megoldás maximumra

A szállítási probléma megfogalmazásánál már beszéltünk arról, hogy a célmátrix lehet költségmátrix, kilométermátrix, nyereségmátrix. Az egyszerű szállítási problémánál a célfüggvényt minimumfeltételre írtuk fel, vagyis minimális költséget, minimális futást kívántunk elérni. Azonban egy cégnek nem lehet célja minimális nyereséget elérni. Tehát ha a célmátrix elemei nyereséget tartalmaznak, és célunk a maximális nyereség elérése, az eddigiekben ismertetett módon nem oldható meg ez a probléma. Annak érdekében, hogy a feladatot mégis meg tudjuk oldani, az eljárást módosítani kell.

A 2.1. táblázat adatait felhasználva tételezzük fel, hogy most a c_{ij} elemek nyereséget fejeznek ki. Célunk nem lehet más, mint a maximális nyereség elérése. Mielőtt a feladat részletezéséhez kezdenénk, nézzük meg, hogyan írható fel általánosságban a maximumfeladat:

(2.17) $$x_{ij} \geq 0,$$

ahol $1 \leq i \leq m$, $0 \leq j \leq n$.

(2.18) $$\sum_{j=1}^{n} x_{ij} = a_i$$

(2.19) $$\sum_{i=1}^{m} x_{ij} = b_j$$

(2.20) $$\sum_{j=1}^{n} b_j = \sum_{i=1}^{m} a_i$$

(2.21) $$C = \sum_{i,j} c_{ij} x_{ij} \quad \rightarrow \quad \max$$

Összehasonlítva az előzőeket az egyszerű szállítási problémánál felírt matematikai összefüggésekkel, eltérést csak a célfüggvénynél tapasztalunk. Fontos azonban megjegyezni, hogy míg az egyszerű szállítási problémánál a célmátrix elemei csak nemnegatív számok lehetnek, azaz $c_{ij} \geq 0$ minden elemre igaz volt, hiszen negatív távolságokról, illetve negatív költségről nem beszélhetünk, addig a maximumfeladatnál a célmátrix elemei negatív értékűek is lehetnek, mivel előfordulhat „negatív nyereség", vagyis veszteség is. A feladat kétféleképpen is megoldható.

1. A megoldás menete alapvetően megegyezik a már ismertetett eljárással. Az eltérés az alábbiakban van:

- Az induló program készítésénél mindig a legnagyobb elemre programozunk.
- A program akkor javítható, ha a nem kötött c_{ij} elemekhez tartozó differenciák között van pozitív. Azt az elemet célszerű bevonni a programba, amelyre a legnagyobb pozitív differenciát kapjuk.

2. Feladatunkat egy másik módszerrel is megoldhatjuk. Ennek során kihasználunk egy tételt, amely szerint valamely célfüggvény maximuma azonos a célfüggvény (–1)-szeresének minimumával, vagyis:

(2.22) $$\sum_{i,j} -c_{ij} x_{ij} \rightarrow \min \quad = \quad \sum_{i,j} c_{ij} x_{ij} \rightarrow \max$$

Bizonyítás nélkül elfogadva ezt a tételt láthatjuk, hogy ha a célmátrix minden egyes elemét megszorozzuk (–1)-gyel, akkor a feladatot minimumfeladatként oldhatjuk meg (a célfüggvény x_{ij} elemei csak nemnegatív számok lehetnek). A megoldás során azonban ügyelni kell arra, hogy a célmátrixot a (–1)-gyel történő szorzás után mindig nullára kell redukálni, mégpedig úgy, hogy minden egyes elemünk nulla vagy pozitív szám legyen. Ezt úgy érhetjük el, ha minden sorból és oszlopból a legnegatívabb számot vonjuk le.

Nézzük meg a 2.1. táblázatban bemutatott példán keresztül, hogyan kell a maximumfeladatot így megoldani. (–1)-gyel való szorzás (2.28. táblázat):

2.28. táblázat

	F_1	F_2	F_3	F_4	F_5	
K_1	–6	–2	–8	–7	–5	40
K_2	–4	–3	–7	–5	–9	70
K_3	–2	–1	–3	–6	–4	60
K_4	–5	–6	–4	–8	–3	30
	30	60	50	40	20	200

A nullára redukálást először soronként (sorminimumok: –8, –9, –6, –8), majd oszloponként végrehajtva (oszlopminimumok: 2, 2, 0, 0, 0), a 2.29. táblázatot kapjuk.

2.29. táblázat

	F_1	F_2	F_3	F_4	F_5	
K_1	0	4	0	1	3	40
K_2	3	4	2	4	0	70
K_3	2	3	3	0	2	60
K_4	1	0	4	0	5	30
	30	60	50	40	20	200

A kapott mátrix alapján már megoldhatjuk a példát a minimumfeladatnál ismertetett módszerrel.

2.4. A Vogel–Korda-módszer

A szállítási probléma disztribúciós módszerrel történő megoldása minden esetben optimális megoldást ad. Nagyobb mátrixok esetében a megoldás menete azonban igen hosszadalmas lehet, ami a módszer gyakorlati alkalmazásának korlátot szab. Ezért ilyen esetekben érdemesebb olyan módszert alkalmazni, amely gyorsabban szolgáltat megoldást, de ugyanakkor az optimumtól nem tér el jelentős mértékben. Erre a problémára igen hatékony módszert dolgoztak ki Nyles V. Reinfeld és William R. Vogel amerikai matematikusok 1958-ban[7], amelyet később, 1967-ben Benedikt Korda cseh

[7] Reinfeld, Nyles V. – Vogel, William R. (1958): *Mathematical Programming*. Prentice-Hall, Englewood Cliffs.

matematikus tökéletesített[8]. Róluk nevezték el az eljárást Vogel–Korda-féle közelítő eljárásnak (*Vogel's Approximation Method*, VAM). A módszer lényegét és gyakorlati alkalmazását egy konkrét példa megoldásán keresztül mutatjuk be.

A feladatot itt is lehet nullára redukálással kezdeni, bár ez nem feltétlenül szükséges. Most is a 2.1. pontban ismertetett példát oldjuk meg, és a redukált mátrixot írjuk fel (2.30. mátrix).

2.30. mátrix

$$\begin{bmatrix} 3 & 0 & 4 & 1 & 1 \\ 2 & 2 & 4 & 0 & 6 \\ 0 & 0 & 0 & 1 & 1 \\ 3 & 5 & 1 & 3 & 0 \end{bmatrix}$$

A következő lépésben rangsorolni kell a mátrix oszlopait és sorait annak érdekében, hogy meg tudjuk határozni, mely sorban vagy oszlopban kell kezdeni a programozást. A rangsorolást az úgynevezett büntetések *(penalty)* segítségével hajtjuk végre. A büntetéseket úgy képezzük, hogy a redukált mátrix minden sorából és oszlopából kiválasztjuk a két legkisebb számot (amelyek lehetnek azonosak is), majd kivonjuk azokat egymásból, mindig a nagyobból a kisebbet.

A K_1 sor büntetése tehát a sor két legkisebb elemének (0 és 1) különbségéből adódik, értéke 1 lesz. Ugyanígy járunk el a többi sorban és oszlopban is. A kapott értékek lesznek a büntetések, amelyeket a 2.31. táblázat szemléltet.

2.31. táblázat

	F_1	F_2	F_3	F_4	F_5		
K_1	3	0	4	1	1	40	1
K_2	2	2	4	0	6	70	2
K_3	0	0	0	1	1	60	0
K_4	3	5	1	3	0	30	1
	30	60	50	40	20	200	
	2	0	1	1	1		büntetések

[8] Korda, Benedikt (1967): *Matematické metody v ekonomii*. Praha, SNTL.

Mint már említettük, a büntetések alapján döntjük el, melyik sorban, illetve oszlopban kezdjük a programozást. Ott indulunk el, ahol a büntetés a legnagyobb. Ezért a módszert a legnagyobb különbségek módszerének is szokták nevezni.

A programba az a viszonylat vonható be, amely az alábbi feltételeknek eleget tesz:

- legnagyobb büntetéssel rendelkező sorban vagy oszlopban található,
- ebben a sorban vagy oszlopban a legkisebb költségelem,
- a legkisebb költségelemre a legnagyobb mennyiséget tudjuk programozni.

Ha ez a három feltétel több elemnél egyszerre teljesül, akkor tetszőleges, hogy melyik helyet választjuk indulásnak.

Példánkban az első két feltétel a K_2 sorban és az F_1 oszlopban egyszerre teljesül, mivel mindkettő büntetése 2, a legkisebb költségelem pedig a c_{24}, illetve a c_{31}, mindkettő 0 értékkel. Meg kell tehát vizsgálni, hogy a harmadik feltételnek melyik hely tesz eleget, hol kezdhetjük a programozást. A c_{31} elemre csak 30 rakományegységet programozhatunk, mivel az F_1 fogadóhely igénye maximum ennyi. A c_{24} elemre viszont már 40 rakományegység programozható, tehát ezen a helyen kezdjük el a programozást. Az első lépést a 2.32. táblázat szemlélteti.

2.32. táblázat

	F_1	F_2	F_3	F_4	F_5		
K_1	3	0	4	1	1	40	1
K_2	2	2	4	0 40	6	~~70~~ 30	2
K_3	0	0	0	1	1	60	0
K_4	3	5	1	3	0	30	1
	30	60	50	40	20		
	2	0	1	1	1		

Látható, hogy a c_{24} elemre 40 rakományegység került, így az F_4 fogadóhely igényét teljes mértékben kielégítettük. A feladat további menetében az F_4 oszlopot úgy tekintjük, mintha nem is szerepelne. A könnyebb áttekinthetőség kedvéért a gyakorlatban át is szoktuk húzni.

Következő lépésként ismét képezni kell a büntetéseket, sor kiesése esetén az oszlopokban, oszlop kiesése esetén a sorokban kapunk új büntetéseket (2.33. táblázat), de a különbségek megállapításánál nem szabad figyelembe venni a kiesett sort vagy oszlopot.

2.33. táblázat

	F_1	F_2	F_3	F_4	F_5		
K_1	3	0	4	1	1	40	1
K_2	2	2	4	[0]40	6	30	0
K_3	0	0	0	1	1	60	0
K_4	3	5	1	3	0	30	1
	30	60	50		20		
	2	0	1		1		

Az új büntetések meghatározása után a programozást a már ismertetett módon folytatjuk. Megkeressük a legnagyobb büntetéshez tartozó sort vagy oszlopot (F_1 oszlop), majd ebben a legkisebb elemet (c_{31}), és ráprogramozzuk a lehető legnagyobb mennyiséget, 30 rakományegységet. Most kiesett a programból az F_1 oszlop, mivel az összes igényét kielégítettük (2.34. táblázat).

2.34. táblázat

	~~F_1~~	F_2	F_3	F_4	F_5		
K_1	~~3~~	0	4	1	1	40	1
K_2	~~2~~	2	4	[0]40	6	30	0
K_3	[0]30	0	0	1	1	~~60~~ 30	0
K_4	~~3~~	5	1	3	0	30	1
	~~30~~	60	50		20		
	~~2~~	0	1		1		

Újból meghatározzuk a büntetéseket, amelyeknél most csak a sorokban van változás, és mindaddig folytatjuk a programozást, míg minden igényt ki nem elégítettünk.

A harmadik lépésben a második sor büntetése (2) a legnagyobb. A sorban a c_{22} elem értéke a legkisebb (2), így erre programozzuk a lehető legnagyobb mennyiséget. Az F_2 fogadóhely igénye 60 rakományegység, de a K_2 kibocsátóhely kapacitása már csak 30 egység. Így a c_{22} elemre maximum 30

egység programozható. Ezzel a K_2 kibocsátóhely ki is esik a további programozásból, mivel az összes kapacitását elprogramoztuk (2.35. táblázat).

2.35. táblázat

	F_1	F_2	F_3	F_4	F_5		
K_1	3	0	4	1	1	40	1
~~K_2~~	~~2~~	~~2~~ 30	~~4~~	~~0~~ 40	~~6~~	~~30~~	~~2~~
K_3	0 30	0	0	1	1	30	0
K_4	3	5	1	3	0	30	1
		~~60~~ 30	50		20		
		0	1		1		

Az oszlopokban újból büntetéseket számolunk, látható azonban, hogy most nem történt változás. A negyedik lépésben az első feltételnek két sor és két oszlop is eleget tesz, sőt mind a sorokban, mind az oszlopokban a legkisebb elem nulla. Meg kell tehát vizsgálni a harmadik feltételt is. A c_{33} elemre 30, a c_{45}-re 20, a c_{12}-re szintén 30 rakományegység programozható. Tehát a c_{33} és a c_{12} elemre minden feltétel azonos, a választás tehát tetszőleges. Programozzuk a c_{12} elemre a 30 rakományegységet. Ezzel a további programozásból kiesik a második oszlop is (2.36. táblázat).

2.36. táblázat

	F_1	F_2	F_3	F_4	F_5		
K_1	3	0 30	4	1	1	~~40~~ 10	1
~~K_2~~	~~2~~	~~2~~ 30	~~4~~	~~0~~ 40	~~6~~		
K_3	0 30	0	0	1	1	30	0
K_4	3	5	1	3	0	30	1
		~~30~~	50		20		
		0	1		1		

Most soronként kell új büntetéseket számolnunk. A legnagyobb értékűt az első sorban kapjuk. Itt a legkisebb elem a c_{15}, és maximális 10 egység programozható rá, mivel a K_1 kibocsátóhely kapacitásából 30 egységet már elprogramoztunk. Ezzel az első sor is kiesett (2.37. táblázat).

2.37. táblázat

	F_1	F_2	F_3	F_4	F_5		
K_1	3	0 [30]	4	1	1 [10]	10	3
K_2	2	2 [30]	4	0 [40]	6		
K_3	0 [30]	0	0	1	1	30	1
K_4	3	5	1	3	0	30	1
			50		20 10		
			1		1		

Oszloponként új büntetéseket számolunk, de értékük nem fog változni. Minden sorban és minden oszlopban a költségmátrix legkisebb értéke is megegyezik (0). A harmadik feltétel alapján a c_{33} elemet vonjuk be a programba, mert ide tudjuk a legnagyobb mennyiséget (30) programozni. A K_3 kibocsátóhely kapacitását teljes mértékben elprogramoztuk, így az is kiesik a programból (2.38. táblázat).

2.38. táblázat

	F_1	F_2	F_3	F_4	F_5		
K_1	3	0 [30]	4	1	1 [10]		
K_2	2	2 [30]	4	0 [40]	6		
K_3	0 [30]	0	0 [30]	1	1	30	1
K_4	3	5	1	3	0	30	1
			50 20		10		
			1		1		

További büntetéseket már nem kell számolni, mivel csak két viszonylatban történhet szállítás, a többi relációt már kizártuk. Tehát a K_4 kibocsátóhely kapacitásából a c_{43} elemre 20-at, a c_{45} elemre 10 rakományegységet programozunk. Ezzel a fogadóhelyek igényeit teljes mértékben kielégítettük, és az összes kapacitásunkat felhasználtuk.

A feladat végleges megoldását, összefoglalva az eddig részletezve bemutatott lépéseket, a 2.39. táblázat tartalmazza. A büntetések fölött lévő bekarikázott számok a programozás lépéseit mutatják.

2.39. táblázat

	F_1	F_2	F_3	F_4	F_5		① ② ③ ④ ⑤ ⑥
K_1	3	$\boxed{0}^{30}$	4	1	$\boxed{1}^{10}$	40	1 1 1 1 3 –
K_2	2	$\boxed{2}^{30}$	4	$\boxed{0}^{40}$	6	70	2 0 2 – – –
K_3	$\boxed{0}^{30}$	0	$\boxed{0}^{30}$	1	1	60	0 0 0 0 1 1
K_4	3	5	$\boxed{1}^{20}$	3	$\boxed{0}^{10}$	30	1 1 1 1 1 1
	30	60	50	40	20	200	
①	2	0	1	1	1		
②	2	0	1	–	1		
③	–	0	1	–	1		
④	–	0	1	–	1		
⑤	–	–	1	–	1		
⑥	–	–	1	–	1		

A program értékére

$$C = C_R + 0\cdot30 + 1\cdot10 + 2\cdot30 + 0\cdot40 + 0\cdot30 + 0\cdot30 + 1\cdot20 + 0\cdot10 =$$
$$= 570 + 90 = 660 \text{ km}$$

adódik. Ez 20 km-rel, vagyis 3%-kal több az optimális 640 km-nél. A Vogel–Korda-módszer egyszerűbb feladatoknál gyakran azonnal az optimumot adja, nagyobb mátrixok esetében pedig a korábban bemutatott javító eljárásokkal érhetjük el az optimális értéket.

Vákát oldal

3. A hozzárendelési probléma

Legyen adott *m* darab kibocsátóhely, egyenként 1 egységnyi kapacitással, *m* darab fogadóhely, mind 1 egységnyi igénnyel, valamint az ezek közti kapcsolatokat leíró célmátrix. A hozzárendelési probléma megoldása annak meghatározását jelenti, hogy melyik kibocsátóhely-fogadóhely relációt vegyük igénybe a program szélsőértékének elérése érdekében. Ez a diszpozíciós mátrix szempontjából azt jelenti, hogy az egyes x_{ij} értékek vagy 0, vagy 1 lesznek, és mindegyik sorban és mindegyik oszlopban pontosan egy darab 1 értékű elem lesz.

Ha a célmátrix a kibocsátó- és fogadóhelyek közti távolságokat vagy költségeket tartalmazza, akkor a hozzárendelési probléma a szállítási probléma egy speciális esete, amikor nem azt kell meghatároznunk, melyik kibocsátóhelyről melyik fogadóhelyre mennyi árut kell elszállítani (hiszen ez adott, 1 egységnyit), hanem a kibocsátóhelyeket és a fogadóhelyeket úgy kell párba állítanunk, hogy a megfelelő c_{ij} elemek összege minimális legyen. Ezt a párba állítást nevezzük minimális súlyú teljes párosításnak.

Ha a célmátrix például nyereséget ír le, akkor a párba állítás abban az esetben optimális, ha a megfelelő c_{ij} elemek összege maximális, ezt nevezzük maximális súlyú teljes párosításnak.

A problémát 1955-ben oldotta meg Harold W. Kuhn amerikai matematikus két magyar matematikus eredményeinek felhasználásával.[9] 1936-ban jelent meg ugyanis Lipcsében németül Kőnig Dénesnek *A véges és végtelen gráfok elmélete*[10] című műve, amely az első gráfelméletről írott könyv. Ennek angol nyelvű kiadását olvasva figyelt fel Kuhn egy lábjegyzetre, amely Egerváry Jenő 1931-ben a *Matematikai és Fizikai Lapok*ban magyarul megjelent cikkére[11] utalt. Kuhn beszerezte a cikk egy másolatát, és egy

---

[9] Kuhn, Harold W. (1955): The Hungarian Method for the Assignment Problem. *Naval Res. Logist. Quart.,* No. 2. 83–97.; Kuhn, Harold W. (1956): Variants of the Hungarian Method for Assignment Problems. *Naval Res. Logist. Quart.,* No. 3. 253–258.

[10] König, Dénes (1936): *Theorie der endlichen und unendlichen Graphen.* Leipzig, Akademische Verlagsgesellschaft. 254.

[11] Egerváry Jenő (1931): Mátrixok kombinatorikus tulajdonságairól. *Matematikai és Fizikai Lapok,* 38. sz. 16–28.

magyar szótár és egy nyelvtankönyv segítségével (!) lefordította a cikket. Ezen két forrás segítségével dolgozta ki eljárását, amelyet az ötletadók iránti tiszteletből nevezett el magyar módszernek. Ezen a néven is publikálta, így a szakirodalomban is a *Hungarian method* elnevezéssel terjedt el.

Nézzük most meg, hogyan lehet megfogalmazni a feladatot általános matematikai formulával:

(3.1) $$x_{ij} \in \{0,1\},$$

ahol $1 \le i \le m$, $0 \le j \le m$.

(3.2) $$\sum_{i=1}^{m} x_{ij} = \sum_{j=1}^{m} x_{ij} = 1$$

(3.3) $$C = \sum_{i,j} c_{ij} x_{ij} \quad \rightarrow \quad \min$$

A hozzárendelési probléma magyar módszerrel való megoldásának menetét részletesen egy példán keresztül fogjuk bemutatni.

3.1. Az induló program

Tételezzük fel, hogy öt különböző kibocsátóhelyről kell szállítani öt különböző fogadóhelyre. Határozzuk meg az irányítási programot, vagyis: melyik kibocsátóhelyről melyik fogadóhelyre szállítsunk, ha minimális összfutást akarunk elérni a szállítás során. A kibocsátó- és a fogadóhelyek egymástól mért távolságát a 3.1. táblázat szemlélteti.

3.1. táblázat

	F_1	F_2	F_3	F_4	F_5	
K_1	5	3	4	2	6	1
K_2	8	3	5	5	4	1
K_3	2	5	3	6	8	1
K_4	4	2	8	3	6	1
K_5	3	6	9	5	3	1
	1	1	1	1	1	

A hozzárendelési probléma megoldását minden esetben a célmátrix nullára redukálásával kell kezdeni, ellentétben a szállítási problémával, ahol ez csak opcionális. Erre azért van szükség, mert a feladat megoldása során csak a nulla elemekre szabad majd programozni.

Redukáljunk először soronként nullára. Válasszuk ki a sorok legkisebb elemeit, jelöljük ezeket u_i-vel, majd képezzünk azok összegét (3.2. táblázat). A $\sum u_i$ értékek jelentőségével később foglalkozunk.

3.2. táblázat

	F_1	F_2	F_3	F_4	F_5		u_i
K_1	5	3	4	2	6	1	2
K_2	8	3	5	5	4	1	3
K_3	2	5	3	6	8	1	2
K_4	4	2	8	3	6	1	2
K_5	3	6	9	5	3	1	3
	1	1	1	1	1		$\sum u_i = 12$

Ezt követően válasszuk ki a már soronként redukált mátrix oszlopainak legkisebb elemeit, jelöljük azokat v_j-vel, majd képezzük a $\sum v_j$ összeget is (3.3. táblázat).

3.3. táblázat

	F_1	F_2	F_3	F_4	F_5	
K_1	3	1	2	0	4	1
K_2	5	0	2	2	1	1
K_3	0	3	1	4	6	1
K_4	2	0	6	1	4	1
K_5	0	3	6	2	0	1
	1	1	1	1	1	
v_j	0	0	1	0	0	$\sum v_j = 1$

Most már minden sorban és oszlopban van nulla elem, így elkezdhetjük az induló program elkészítését. A programozást soronként kezdjük. Megkeressük a független nulla elemeket, azaz azokat a 0 értékű szabad c_{ij} elemeket, amelyek sorában nem található több szabad 0 értékű elem,

és ezekhez rendeljük hozzá a szállításokat. Ez praktikusan azt jelenti, hogy a megfelelő x_{ij} elem értékét 1-gyé tesszük. Az első sorban máris találunk ilyen elemet, c_{14}-et, ezért erre programozunk. Tehát az F_4 fogadóhelyre a K_1-es kibocsátóhelyről szállítunk. Ezzel a további programozásból az első sort és a negyedik oszlopot ki is zárhatjuk, mivel sem erről a kibocsátóhelyről nem lehet többet elszállítani, sem a fogadóhely nem igényel már többet. Nem kell az egész sort vagy oszlopot kihúzni, elegendő csak a nulla elemeket áthúzni. Esetünkben sem az oszlopban, sem a sorban nincs több nulla elem (3.4. táblázat).

3.4. táblázat

	F_1	F_2	F_3	F_4	F_5	
K_1	3	1	1	[0]	4	1
K_2	5	0	1	2	1	1
K_3	0	3	0	4	6	1
K_4	2	0	5	1	4	1
K_5	0	3	5	2	0	1
	1	1	1	1	1	

Folytassuk a programozást a második sorban. Itt a c_{22} elem a független nulla, tehát ide programozunk. Mivel a második sor és oszlop is kiesik, így a második oszlopban lévő 0 elemet (c_{42}-t) áthúzzuk, vagyis kizárjuk a további programozásból (3.5. táblázat).

3.5. táblázat

	F_1	F_2	F_3	F_4	F_5	
K_1	3	1	1	[0]	4	1
K_2	5	[0]	1	2	1	1
K_3	0	3	0	4	6	1
K_4	2	~~0~~	5	1	4	1
K_5	0	3	5	2	0	1
	1	1	1	1	1	

A harmadik sorban két nulla elem is van, amelyekre még nem programoztunk, sem át nem húztuk, azaz egyik sem független nulla, így nem tudjuk egyértelműen eldönteni, melyikre programozzunk. Ha a további programozás

során majd egyetlen független nulla elemet sem találunk, akkor a több nulla elem közül tetszőlegesen választhatunk. Ebben az esetben a programunk alternatív optimumot ad.

Nézzük most a negyedik sort. Itt sem tudunk programozni, hiszen a c_{42} helyen lévő 0 elemet már kizártuk a programozásból. Következik az ötödik sor. Itt ismételten előáll a harmadik sornál már megismert szituáció, vagyis nincs független nulla. Mivel soronként már nem tudunk tovább programozni, keressük meg oszloponként a független nullákat és folytassuk ott a programozást.

Az első oszlopban nincs független nulla, a második oszlopot pedig már kizártuk a programozásból. A harmadik oszlopban viszont találunk független nulla elemet, c_{33}-at, így ide programozhatunk. Ezzel a harmadik sor és oszlop is kiesik, tehát a c_{31} helyen levő nulla elemet áthúzzuk.

A negyedik oszlop már szintén kiesett a programozásból. Marad tehát az ötödik oszlop, ahol ismételten találunk független nulla elemet, c_{55}-öt, amire programozhatunk. Most az ötödik sor és oszlop kiesése miatt a c_{51} helyen levő nulla elemet kell áthúzni.

A programozást tovább nem tudjuk folytatni, mivel nincs több szabad nulla elemünk: amire lehetett, programoztunk, a többit pedig áthúztuk, azaz kizártuk a programozásból.

Az induló programot tehát elkészítettük, azonban ha jól megfigyeljük az ezt bemutató 3.6. táblázatot, láthatjuk, hogy öt viszonylat helyett csak négyet sikerült programoznunk, vagyis egy fogadóhely (F_1) igényét nem elégítettük ki. Jelöljük a kielégítetlen igények számát *h*-val, azaz jelen esetben $h_1 = 1$, ahol az alsó indexbeli 1 azt mutatja, hogy ez az első programozás végén maradt kielégítetlen igények száma. Erre a későbbiekben még szükségünk lesz.

3.6. táblázat

	F_1	F_2	F_3	F_4	F_5	
K_1	3	1	1	[0]	4	1
K_2	5	[0]	1	2	1	1
K_3	~~0~~	3	[0]	4	6	1
K_4	2	~~0~~	5	1	4	1
K_5	~~0~~	3	5	2	[0]	1
	1	1	1	1	1	

3.2. A program javítása

Hogy minden fogadóhely igényét kielégítsük, az induló programot javítanunk kell. A javítást úgynevezett fedővonalak húzásával kezdjük. Jelöljük meg a programba be nem vont sorokat és oszlopokat egy csillaggal, majd a megjelölt sorokban és oszlopokban található nulla elemekre merőlegesen húzzunk fedővonalakat.

Példánkban csak egy sor (K_4) és egy oszlop (F_1) marad szabadon, így ezeket kell csillaggal megjelölni. A K_4 sorban egy darab 0 elem található, a c_{42}, így a K_4 sorra merőlegesen ebben az elemben húzunk fedővonalat, azaz az F_2 oszlopot fedjük le. Az F_1 oszlopban két nulla elem található, c_{31} és c_{51}, azaz az ezekre merőleges sorokat, a K_3 és a K_5 sorokat fedjük le. Nem sikerült azonban minden nulla elemet lefedni. Ezért a szabadon maradt 0 eleme(ke)t tetszés szerint választott fedővonalakkal kell áthúzni, betartva két igen fontos szabályt:

- minden nulla elemet le kell fedni,
- kötött elemen fedővonalak nem metszhetik egymást.

Ezeket figyelembe véve a még le nem fedett c_{14} nulla elemre akár függőlegesen, akár vízszintesen húzhatunk fedővonalat. Példánkban fedjük le ezt az elemet a K_1 sor áthúzásával. Az így kialakított fedővonalrendszert a 3.7. táblázatban láthatjuk.

3.7. táblázat

	F_1	F_2	F_3	F_4	F_5		
K_1	~~3~~	~~1~~	~~1~~	~~0~~	~~4~~	~~1~~	
K_2	5	~~0~~	1	2	1	1	
K_3	~~0~~	~~3~~	~~0~~	~~4~~	~~6~~	~~1~~	
K_4	2	~~0~~	5	1	4	1	*
K_5	~~0~~	~~3~~	~~5~~	~~2~~	~~0~~	~~1~~	
	1	~~1~~	1	1	1		
	*						

Az úgynevezett Kőnig–Egerváry-tétel azt mondja ki, hogy a független nullák maximális száma megegyezik a fedővonalak minimális számával a hozzárendelési probléma megoldása során. Független nulláknak azokat a nulla elemeket nevezzük, amelyekre programoztunk. Példánkban

négy ilyen elem található, amit nevezzünk maximális számnak. A mátrixban található összes nulla elemet szintén négy vonallal tudtuk lefedni. Ennél kevesebbel nem lehet az összes nulla elemet lefedni. Ez minimális szám, hiszen több fedővonalat is húzhattunk volna. Tehát teljesül a tétel, így folytathatjuk a programozást.

A program javítása az úgynevezett ε-transzformáció segítségével történik. Már említettük, hogy a hozzárendelési problémánál csak nulla elemekre programozhatunk, tehát az a célunk, hogy növeljük a nulla elemek számát a mátrixban, lehetővé téve így mind az öt igény elprogramozását.

Az ε-transzformációt a következőképpen hajtjuk végre.

- A le nem fedett elemek közül kiválasztjuk a legkisebbet, példánkban ez a c_{23} elem, ez lesz az ε-transzformáció során az epszilon: $\varepsilon_1 = 1$, ahol az alsó indexbeli 1 azt mutatja, hogy ez az első javításhoz tartozó ε.
- ε_1-et levonjuk minden le nem fedett elemből,
- ε_1-et hozzáadjuk minden lefedett elemhez,
- az egyszer lefedett elemeket változatlanul leírjuk.

Betartva ezeket a szabályokat a 3.8. táblázatot kapjuk az első ε-transzformáció eredményeként.

3.8. táblázat

	F_1	F_2	F_3	F_4	F_5	
K_1	3	2	1	0	4	1
K_2	4	0	0	1	0	1
K_3	0	4	0	4	6	1
K_4	1	0	4	0	3	1
K_5	0	4	5	2	0	1
	1	1	1	1	1	

Az új mátrixot felhasználva elvégezzük a programozást a már ismertetett módon. A c_{14} és a c_{42} elemre minden további nélkül tudunk programozni. Ezt követően viszont nincs több független nulla elem, viszont vannak még szabad nullák, így ezek közül tetszőlegesen választhatunk. Harmadik lépésben programozzunk tehát például a c_{23} elemre. A továbbiakban már egyértelműen lehet programozni a következő sorrendben: c_{31}, c_{55} (3.9. táblázat).

3.9. táblázat

	F_1	F_2	F_3	F_4	F_5	
K_1	3	2	1	[0]	4	1
K_2	4	~~0~~	[0]	1	~~0~~	1
K_3	[0]	4	~~0~~	4	6	1
K_4	1	[0]	4	~~0~~	3	1
K_5	~~0~~	4	5	2	[0]	1
	1	1	1	1	1	

Minden sorban és oszlopban van kötött elem, tehát a programunk optimális. Határozzuk meg a program értékét először az eredeti mátrix elemeinek felhasználásával, majd ellenőrizzük a megoldás helyességét:

$C = 2 + 5 + 2 + 2 + 3 = 14$ km.

Tehát 14 km szükséges feltétlenül ahhoz, hogy a szállítást el tudjuk végezni. A kapott eredmény helyességét a redukálás során meghatározott $\sum u_i$ és $\sum v_j$ értékek, valamint az egyes javításokhoz tartozó ε és h értékek segítségével ellenőrizhetjük. Ha összeadjuk a sor- és oszlopminimumok összegét, $\sum u_i$-t és $\sum v_j$-t, valamint az egyes javításoknál kapott ε-ok és h-k szorzatait minden egyes javításra, a célfüggvény értékét kell kapnunk:

$$C = \sum_{i=1}^{m} u_i + \sum_{i=1}^{m} v_j + \sum_{k=1}^{r} \varepsilon_k h_k , \tag{3.4}$$

ahol r a javítások száma, h_k a h-adik javításban ki nem elégített igények száma és ε_k a k-adik javításban használt ε-érték. Esetünkben tehát ezen a módon kiszámítva a program értékét kapjuk:

$C = 12 + 1 + 1 \cdot 1 = 14.$

A két módon kapott érték megegyezik, tehát jól számoltunk.

4. A szállítási probléma megoldása magyar módszerrel

A Kőnig–Egerváry-tételt nemcsak a hozzárendelési probléma, hanem a szállítási probléma megoldásában is kihasználhatjuk.

Oldjuk meg a 2.1. pontban ismertetett feladatot magyar módszerrel (4.1. táblázat). A szállítási probléma általános ismertetésénél említettük, hogy az igényeket és a kapacitásokat egységenként kell értelmezni. Ebből kiindulva a mátrixunkat felfoghatjuk úgy is, mintha az 200 sorból és oszlopból állna. Ebben az esetben hozzárendelési problémaként oldhatjuk meg. Ilyen nagy méretű mátrixszal azonban igen körülményes számolni, és mint látni fogjuk, nincs is rá szükség, mivel a mátrix igényei és kapacitásai úgy is tekinthetők, mint az egység többszörösei. Ezért is nevezzük az egyes sorok kapacitásait, illetve az egyes oszlopok igényeit multiplicitásnak.

4.1. táblázat

	F_1	F_2	F_3	F_4	F_5	a_i
K_1	6	2	8	7	5	40
K_2	4	3	7	5	9	70
K_3	2	1	3	6	4	60
K_4	5	6	4	8	3	30
b_j	30	60	50	40	20	200

4.1. Az induló program

A feladat megoldását most is nullára redukálással kezdjük. Vigyázni kell azonban, hogy mind a sor-, mint az oszlopminimumok összeadásánál figyelembe kell venni a multiplicitásokat is.

Példánkban a sorminimumok értéke és összege a következő:

$u_1 = 2$

$u_2 = 3$

$u_3 = 1$
$u_4 = 3$
$u_1 a_1 = 2 \cdot 40 = 80$
$u_2 a_2 = 3 \cdot 70 = 210$
$u_3 a_3 = 1 \cdot 60 = 60$
$u_4 a_4 = 3 \cdot 30 = 90$
$\sum u_i a_i = 440$

A soronkénti redukálás után a 4.2. táblázatot kapjuk.

4.2. táblázat

	F_1	F_2	F_3	F_4	F_5	a_i
K_1	4	0	6	5	3	40
K_2	1	0	4	2	6	70
K_3	1	0	2	5	3	60
K_4	2	3	1	5	0	30
b_j	30	60	50	40	20	200

Az oszloponkénti redukálásnál és összegzésnél a már ismertetett módon járunk el.

$v_1 = 1$
$v_2 = 0$
$v_3 = 1$
$v_4 = 2$
$v_5 = 0$
$v_1 b_1 = 1 \cdot 30 = 30$
$v_2 b_2 = 0 \cdot 60 = 0$
$v_3 b_3 = 1 \cdot 50 = 50$
$v_4 b_4 = 2 \cdot 40 = 80$
$v_5 b_5 = 0 \cdot 20 = 0$
$\sum v_j b_j = 160$

A nullára redukálás után már minden sorban és oszlopban található nulla elem (4.3. táblázat), így elkezdhetjük az induló program elkészítését. A programozást, mint azt már láttuk, soronként kezdjük, és lehetőség szerint arra a nulla elemre programozzunk, amelyikre ezt egyértelműen tehetjük.

4.3. táblázat

	F_1	F_2	F_3	F_4	F_5	a_i
K_1	3	0	5	3	3	40
K_2	0	0	3	0	6	70
K_3	0	0	1	3	3	60
K_4	1	3	0	3	0	30
b_j	30	60	50	40	20	200

Az első sorban a c_{12} elemre programozhatunk, mivel a sorban nem található több nulla elem. Programozzuk tehát erre az elemre a lehető legtöbbet. Az adott relációban az igény 60 egység, a rendelkezésre álló kapacitás azonban csak 40, így ezt a mennyiséget programozhatjuk a c_{12} elemre.

Nézzük azonban meg azt, hogy az első sorra merőlegesen a második oszlopban van-e még lehetőség programozni. Látható, hogy a F_2 fogadóhelynek még van 20 egység igénye, és az oszlopban találhatók nulla elemek, amelyekre tudunk majd programozni. Tehát szállítási problémánál, ellentétben a hozzárendelési problémánál tárgyaltakkal, csak abban az esetben kell az adott c_{ij} elemhez tartozó sorban vagy oszlopban a nulla elemeket áthúzni, ha a sor- és az oszlopmultiplicitásokat teljes mértékben elprogramoztuk.

Soronként folytatva a programozást látható, hogy nincs több olyan sor, amelyben csak egy nulla elem található. Tehát most oszloponként haladunk tovább. Az első és a második oszlopban nem, de a harmadikban már van olyan nulla, amelyre egyértelműen programozhatunk, a c_{43} elem. Az igényt és a kapacitást figyelembe véve erre az elemre maximum 30 egység programozható. Nézzük meg, hogy az oszlopra merőleges sorban van-e még nulla elem. Van, c_{45}, és ezt át kell húzni, mivel a sorhoz tartozó kapacitásértéket teljes mértékben elprogramoztuk, tehát ezt a nulla elemet ki kell zárni a további programozásból.

A negyedik oszlopban szintén található független nulla elem, c_{24}, amelyre maximum 40 egységet programozhatunk. Az oszlopra merőleges második sorban még maradt 30 egység kapacitás, így ennek a sornak a nulla elemeit nem zárhatjuk ki a további programozásból.

Oszloponként már nem tudjuk folytatni a programozást, ezért kezdjük el újból soronként megkeresni a független nulla elemeket. Ilyet nem sikerült találni, tehát nézzük meg oszloponként is. Így sem találunk. Ilyen esetben az eljárás a következő (ha még van szabad nulla elem):

- abban a sorban, illetve oszlopban folytatjuk a programozást, ahol a legkevesebb nulla elem található,

- arra a nulla elemre programozunk, amelyikre a lehető legnagyobb mennyiséget lehet.

Ha több elemre azonos a második feltétel is, szabadon választhatunk.

Példánkban minden sorban és oszlopban két-két nulla elem van, tehát a második szabály szerint döntjük el, melyik elemet vonjuk be a programba. Válasszuk a második sort, itt a c_{21} elemre maximum 30 egységet tudunk programozni. Most azonban a sorban és az oszlopban egyszerre kielégítettük az összes igényt, és elprogramoztuk a még meglévő kapacitást. Éppen ezért a sorban, illetve oszlopban található nulla elemeket (c_{22}-t és c_{31}-et) a további programozásból ki kell zárni, azaz áthúzzuk őket.

Már csak egyetlen nulla elemünk maradt, amire programozni lehet, c_{32}. Ide maximum 20 egységet programozhatunk, mivel az F_2 fogadóhelynek nincs már több igénye. Ezzel az induló programot el is készítettük (4.4. táblázat). Vizsgáljuk meg, milyen eredményt értünk el.

4.4. táblázat

	F_1	F_2	F_3	F_4	F_5	
K_1	3	$\boxed{0}^{40}$	5	3	3	~~40~~
K_2	$\boxed{0}^{30}$	~~0~~	3	$\boxed{0}^{40}$	6	~~70~~ ~~30~~
K_3	~~0~~	$\boxed{0}^{20}$	1	3	3	~~60~~ 40
K_4	1	3	$\boxed{0}^{30}$	3	~~0~~	~~30~~
	~~30~~	~~60~~ ~~20~~	~~50~~ 20	~~40~~	20	200

A 4.4. táblázatból leolvasható, hogy a harmadik sorban maradt még 40 egység kapacitás, amit nem tudunk elprogramozni, a harmadik és az ötödik sorban pedig 20-20 egység igény, amit nem elégítettünk ki, ezzel $h_1 = 40$. Egyértelmű tehát, hogy javítani kell a programot.

4.2. A program javítása

A javítást a hozzárendelési problémánál megismert módon hajtjuk végre. Megcsillagozzuk azokat a sorokat, illetve oszlopokat, ahol nem tudunk mindent elprogramozni, majd megalkotjuk a fedővonalrendszert. A K_4 sort és az F_1 és F_2 oszlopokat azért fedtük le, mert csillagos sorban található nullára merőlegesek. Ekkor azonban még van egy lefedetlen nulla elem,

c_{24}. Ezt csak úgy tudjuk lefedni, ha az F_4 oszlopot fedjük le, mivel a K_2 sort nem fedhetjük le, hiszen akkor a c_{21} kötött elemen két fedővonal metszené egymást (4.5. táblázat).

4.5. táblázat

	F_1	F_2	F_3	F_4	F_5		
K_1	3	0 [40]	5	3	3	~~40~~	
K_2	0 [30]	~~0~~	3	0 [40]	6	~~70~~ ~~30~~	
K_3	~~0~~	0 [20]	1	3	3	~~60~~ 40	*
K_4	~~1~~	~~3~~	0 [30]	~~3~~	~~0~~	~~30~~	
	~~30~~	~~60~~ ~~20~~	~~50~~ 20	~~40~~	20	200	
			*		*		

Négy fedővonallal sikerült tehát minden nulla elemet lefedni. Nézzük meg, hogyan érvényesül a Kőnig–Egerváry-tétel a szállítási problémánál. Láthatjuk, hogy ha csak a fedővonalak és a kötött elemek számát vizsgáljuk, a tétel nem érvényesül, hiszen 4 fedővonal és 5 kötött elem van. Azonban szállítási problémánál figyelembe kell venni a multiplicitásokat is. A független elemek számát úgy határozzuk meg, hogy összeadjuk a független elemekre programozott értékeket:

40 + 30 + 40 + 20 + 30 = 160, ami a független elemek maximális száma.

A fedővonalak minimális számának meghatározásánál pedig a fedővonalakkal lefedett sorokhoz és oszlopokhoz tartozó multiplicitásokat kell összeadni:

30 + 30 + 60 + 40 = 160.

Így már érvényesül a tétel, a kötött elemek maximális száma megegyezik a fedővonalak minimális számával.

E kis kitérő után nézzük meg, hogyan javíthatjuk a programot az ε-transzformáció segítségével. A le nem fedett elemek közül válasszuk ki a legkisebbet. Ez a c_{33} elem, tehát az első javításnál $\varepsilon_1 = 1$, a $h_1 \cdot \varepsilon_1$ szorzat pedig 40. Készítsük el az új mátrixot. Az egyszer lefedett elemeket változatlanul hagyjuk, a kétszer lefedettekhez hozzáadjuk, a lefedetlenekből pedig kivonjuk ε_1 értékét (4.6. táblázat).

4.6. táblázat

	F_1	F_2	F_3	F_4	F_5	a_i
K_1	3	0	4	3	2	40
K_2	0	0	2	0	5	70
K_3	0	0	0	3	2	60
K_4	2	4	0	4	0	30
b_j	30	60	50	40	20	200

Ezt követően végezzük el a programozást az ismert módon. A végrehajtás menetét már nem részletezzük, csak a lépések sorrendjét és a programozott értékeket közöljük (4.7. táblázat):

$c_{12} - 40$, $c_{24} - 40$, $c_{45} - 20$, $c_{43} - 10$, $c_{33} - 40$, $c_{21} - 30$, $c_{32} - 20$.

4.7. táblázat

	F_1	F_2	F_3	F_4	F_5	a_i
K_1	3	$\boxed{0}^{40}$	4	3	2	40
K_2	$\boxed{0}^{30}$	$\cancel{0}$	2	$\boxed{0}^{40}$	5	70
K_3	$\cancel{0}$	$\boxed{0}^{20}$	$\boxed{0}^{40}$	3	2	60
K_4	2	4	$\boxed{0}^{10}$	4	$\boxed{0}^{20}$	30
b_j	30	60	50	40	20	200

Tehát sikerült az összes igényt kielégíteni, illetve az összes kapacitást elprogramozni. Határozzuk most meg az optimális program értékét.

Ezt úgy végezhetjük el, hogy az eredeti célmátrix, a 4.1. táblázat megfelelő elemeit megszorozzuk a hozzájuk tartozó programértékekkel:

$C = 2 \cdot 40 + 4 \cdot 30 + 5 \cdot 40 + 1 \cdot 20 + 3 \cdot 40 + 4 \cdot 10 + 3 \cdot 20 = 640$ km.

Tehát optimális irányítási program esetén az összes futás 640 km lesz.

Ellenőrizzük, hogy jól programoztunk-e. Határozzuk meg a minimális kilométerértéket a javításoknál kiszámított h és ε értékek segítségével:

$$C = \sum_{i=1}^{m} a_i u_i + \sum_{i=1}^{n} b_j v_j + \sum_{k=1}^{r} \varepsilon_k h_k. \tag{4.1}$$

$C = 440 + 160 + 40 = 640$ km.

Ezzel a módszerrel is ugyanazt az eredményt kaptuk, tehát a programunk biztosan jó. Meg kell jegyezni, hogy a programnak alternatív megoldása van. Ezt onnan látjuk, hogy a mátrixban található még olyan nulla elem, amelyet nem vontunk be a programozásba: c_{22} és c_{31}.

A szállítási probléma magyar módszerrel történő megoldását egyszerű szállítási problémára mutattuk be, de természetesen felhasználható a módszer összetett szállítási probléma megoldására is, betartva az ott megismert szabályokat.

A magyar módszer előnyei között említhetjük azt is, hogy nem érzékeny a degenerációra, így a feladat megoldása során igen sok kellemetlenségtől óvhat meg bennünket.

Vákát oldal

5. A körutazási probléma

Az 1930-as években merült fel komolyabban, de már az 1830-as években is foglalkoztak azzal a kérdéssel, hogy ha adva van n darab város, amelyek egymástól mért távolságai ismertek, akkor hogyan lehet meghatározni az ezeket pontosan egyszer érintő legrövidebb körutat? A problémára gazdasági vetülete miatt kezdtek el megoldást keresni: egy utazó ügynök (innen a probléma angol neve is, *travelling salesman problem,* TSP) hogyan tudja például egy cég központjából végiglátogatni a különböző városokban található telephelyeket és utána visszatérni a kiindulási helyre úgy, hogy a lehető legrövidebb utat teszi meg?

Kevés számú hely esetén a körüljárási sorrendet egyszerű rátekintéssel vagy az összes lehetséges körút kipróbálásával is meg lehet határozni. Ezt a megoldási „módszert" nevezi az angol *brute force*-nak, nyers erőnek. Ha a kiindulási ponton kívül még (n–1) helyet kell végigjárni, akkor a lehetséges körutak száma (n–1)!/2, hiszen az első meglátogatandó helyet még mind az (n–1) közül választhatjuk, a másodikat már csak (n–2) hely közül választhatjuk ki, és így tovább, míg csak egy marad, ahonnan pedig a kiindulási pontra térünk vissza. A nevezőben a kettes azért szerepel, mert minden útvonal bejárható visszafele is, de azt nem tekintjük különbözőnek. Öt város esetén a *brute force* megoldás még valóban célravezető lehet, hiszen a lehetséges változatok száma 12. Kilenc hely esetén azonban már 20 160 különféle körbejárás lehetséges. Szükség van tehát a feladatnak objektív számításon alapuló meghatározására.

A körutazási probléma megoldására, tekintettel az igen nagyszámú megoldási változatra, gyakorlati feladatokhoz inkább a közelítő, de gyorsabban eredményt adó változatot alkalmazzák.

A feladat megoldásához mindenekelőtt szükség van a d_{ij} célmátrixra, amelynek elemei jelenthetnek távolságot, időt, költséget stb. Amennyiben $d_{ij} = d_{ji}$ (i = 1..n; j = 1..n), akkor szimmetrikus, egyébként aszimmetrikus problémával állunk szemben.

5.1. A korlátozás és szétválasztás módszere

Ez az eljárás a körutazási probléma optimális megoldását adja. Igen hosszadalmas számítások után kapjuk csak meg a végeredményt, ezért a gyakorlati feladatoknál a közelítő eljárásokat helyezik előtérbe. Ennek ellenére célszerű ezzel a megoldási móddal is megismerkedni. A módszert John D. C. Little, Katta G. Murty, Dura W. Sweeney és Caroline Karel dolgozták ki 1963-ban.[12] Eljárásuk az úgynevezett döntési fa módszer.

A feladat megoldását egy konkrét példán keresztül mutatjuk be.

Egy központi raktárból 5 telephely számára kell szállítani. Tételezzük fel, hogy az 5 telephely által igényelt árumennyiséget egy tehergépjárművel képesek vagyunk elszállítani. A központi raktár és a telephelyek közötti távolságadatokat az 5.1. mátrixban adjuk meg. Határozzuk meg a legkevesebb kilométer-felhasználást igénylő körüljárási útvonalat!

5.1. mátrix

$$\begin{bmatrix} A & 25 & 80 & 36 & 40 & 45 \\ 25 & B & 60 & 8 & 35 & 48 \\ 80 & 60 & C & 47 & 50 & 85 \\ 36 & 8 & 47 & D & 26 & 62 \\ 40 & 35 & 50 & 26 & E & 34 \\ 45 & 48 & 85 & 62 & 34 & F \end{bmatrix}$$

Első lépésként meg kell határoznunk a feladat megoldásának azt a legkisebb értékét, amelynél rövidebb körüljárási útvonal nem képzelhető el. Abból a meggondolásból indulunk ki, hogy a kiindulási mátrix redukálásával a bejárás útvonala nem változik, csak az útvonal hossza csökken. Az előbbiekben felírt kiinduló mátrixunkat először sorok szerint, majd oszlopok szerint nullára redukáljuk (5.2. és 5.3. mátrix).

[12] Murty, Katta G. – Karel, Caroline – Little, John D. C. (1962): *The Travelling Salesman Problem: Solution by a method of ranking assignments.* Elérhető: www-personal.umich.edu/~murty/sales.pdf (A letöltés dátuma: 2019. 09. 12.); Little, John D. C. – Murty, Katta G. – Karel, Caroline – Sweeney, Dura W. (1963): An algorithm for the Travel Salesman Problem. *Operations Research,* Vol. 11. No. 6. 972–989.

5.2. mátrix

						sorminimum
A	0	55	11	15	20	25
17	B	52	0	27	40	8
33	13	C	0	3	38	47
28	0	39	D	18	54	8
14	9	24	0	E	8	26
11	14	51	28	0	F	34
						148

5.3. mátrix

A	0	31	11	15	12	
6	B	28	0	27	32	
22	13	C	0	3	30	
17	0	15	D	18	46	
3	9	0	0	E	0	
0	14	27	28	0	F	
oszlopminimum						
11	0	24	0	0	8	= 43

A körút hosszának lehetséges minimális értékét a sor- és oszlopminimumok összege adja meg: 148 + 43 = 191 km.

A redukálás után a mátrix minden sorában és oszlopában találunk nulla elemet. Abban az esetben, ha sikerül minden sorban és oszlopban a nulla elemekre programoznunk, akkor a program értéke megegyezik a sor- és oszlopminimumok összegével, vagyis a körút hosszának alsó határával. Azokban a sorokban, illetve oszlopokban, ahol csak egy nulla elem van, a választás egyértelmű. Ott, ahol több nulla elem van, már megfontolás tárgyát képezi, melyiket vonjuk be a programba. Amennyiben nulla elem valami oknál fogva nem vonható be a programba, az utána következő legkisebb elemet választjuk. Egyértelmű, hogy ebben az esetben a minimális útvonal hossza a bevont elem értékével növekszik. A programozási feladat megkezdése előtt a növekedést minden nulla elemre meghatározzuk, mégpedig úgy, hogy a nulla elemek mellé odaírjuk a vele azonos sorban és oszlopban található következő legkisebb elemek értékeinek összegét. Ezeknek az elemeknek a programba történő bevonása szükségessé válhat, ha a szóban forgó nulla elemre nem tudunk programozni. Nézzük meg, hogyan alakul ez a példánkban. Az AB elem sorában a következő legkisebb elem az AD elem (11), oszlopában pedig a DB elem (0), ezek összege 11, ez kerül az AB elem nullája mellé zárójelben. Elvégezve ezt minden nulla elemre, az 5.4. táblázatot kapjuk.

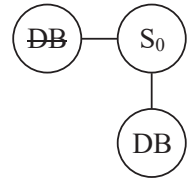
5.4. táblázat

	A	B	C	D	E	F
A	A	0(11)	31	11	15	12
B	6	B	28	0(6)	27	32
C	22	13	C	0(3)	3	30
D	17	0(15)	15	D	18	46
E	3	9	0(15)	0(0)	E	0(12)
F	0(3)	14	27	28	0(3)	F

Elsőként annak a nulla értékű viszonylatnak a körútba vonásának lehetőségét vizsgáljuk, amelynek be nem vonása esetén a legnagyobb lenne a növekmény. Látható, hogy a D-ből B-be, valamint az E-ből C-be vezető útvonalakhoz kapcsolódnak a legnagyobb növekedések, mindkét esetben 15, ezért a két reláció valamelyikénél kell a programozást kezdeni. Mivel a hozzájuk tartozó növekmény azonos, a választás tetszőleges. Példánkban válasszuk induló lépésnek a DB relációt. Mielőtt továbblépnénk, meg kell ismerkednünk néhány jelölési móddal:

- DB jelentse azt, hogy D-ből B-be haladunk,
- ~~DB~~ pedig azt, hogy D-ből B-be biztosan nem megyünk.

A fentiek szerint DB viszonylatnál döntés előtt állunk. Pozitív döntésnek nevezzük, ha egy relációt választunk, negatív döntésnek, ha egy relációt nem választunk. A döntéseket az úgynevezett döntési fán ábrázoljuk, amelynek kiindulási pontját S_0-lal jelöljük. A pozitív döntést a döntési fába függőlegesen rajzoljuk be (például DB: D-ből a B-be megyünk), a negatív döntést pedig vízszintesen (például ~~DB~~: D-ből B-be nem megyünk), lásd 5.1. ábra.



5.1. ábra

Bármelyik megoldást is választjuk, a mátrix bizonyos elemeit többé nem vehetjük figyelembe, hiszen már beszéltünk róla, hogy minden pontot csak egyszer érint a körút. Ha a DB relációt nem választjuk, vagyis a körutat más irányba vezetjük, akkor a DB elem helyére egy kizáró „M”-et írunk.

A minimális program értékét a zárójelbe írt növekménnyel, ami esetünkben 15, megemeljük, majd beírjuk a döntési fába. A megnövelt összeg mutatja, hogy negatív döntés esetén az elérhető legrövidebb útvonal hossza minimálisan mekkora lehet.

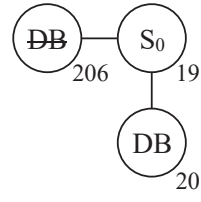
A DB reláció választása esetén szintén töröljük a nulla elemet, de ezzel egy időben a hozzá tartozó sort és oszlopot is töröljük, mivel ezeket többet nem érinthetjük. Ügyelnünk kell arra, hogy azt a relációt, amelyik rövidre zárná a kört, anélkül, hogy valamennyi állomást érinthettünk volna, egy végtelen nagy „M”-mel helyettesítenünk kell, esetünkben ez a BD elem. Példánkban a DB reláció elfogadása esetén a törlések után az 5.5 táblázatot kapjuk.

5.5. táblázat

	A	C	D	E	F
A	A	31	11	15	12
B	6	28	M	27	32
C	22	C	0	3	30
E	3	0	0	E	0
F	0	27	28	0	F

Az 5.5. táblázatot, ha nincs minden sorában és oszlopában nulla elem, tovább kell redukálnunk. A sor- és oszlopminimumok összegével az alsó határ értékét megnöveljük, és beírjuk a döntési fa megfelelő eleméhez. Ez mutatja meg, hogy milyen eredményt hozott a pozitív döntés.

Példánkban az A és a B sorban nincs nulla elem. A sorminimumok értéke 11, illetve 6, így az alsó határ pozitív döntésnél 191 + 11 + 6 = 208 km lesz. A pozitív és a negatív döntés eredményét a döntési fába beírjuk, és azon az ágon haladunk tovább, amelyik az alacsonyabb alsó határral rendelkezik. Ha több ilyen van, akkor azon az ágon, ahol több pozitív döntés van, ha ez is megegyezik, akkor tetszőleges a folytatás. Esetünkben a negatív döntés alsó határa a kisebb (206 km), ezért ezen az ágon haladunk tovább (5.2. ábra).



5.2. ábra

Az eljárást addig folytatjuk, amíg valamely ágon annyi pozitív döntés nincs, ahány csomópontból áll az útvonal. Az alsó határ, ami ennek az ágnak a végén áll, megegyezik a bejárási változat útvonalának hosszával. Ha a többi ágon a kapott értékek magasabbak, megoldásunk optimális. Ha vannak még kisebb vagy egyenlő alsó határral rendelkező, de végig nem vezetett ágak, ezeket folytatni kell egészen addig, amíg vagy jobb megoldást nem kapunk, vagy nyilvánvalóvá nem válik, hogy ezek az ágak rosszabb, esetleg ugyanolyan végeredményt nem adnak.

Mint említettük, példánkban a negatív döntési ágon kell továbbhaladnunk. Az 5.6. táblázat a negatív döntés miatti DB elem „M”-mel való helyettesítése és a redukálás utáni mátrixot mutatja, a nulla elemek mellé írt esetleges növekményekkel együtt.

5.6. táblázat

	A	B	C	D	E	F
A	A	0(20)	31	11	15	12
B	6	B	28	0(6)	27	32
C	22	13	C	0(3)	3	30
D	2	M	0(2)	D	3	31
E	3	9	0(0)	0(0)	E	0(12)
F	0(2)	14	27	28	0(3)	F

A növekmény az AB viszonylatban a legnagyobb. Negatív döntés, azaz ~~AB~~ választása esetén az alsó határ értéke az AB elem melletti zárójeles kilométerértékkel: 206 + 20 = 226 km-re nő. Pozitív döntés esetén az 5.7. táblázatot kapjuk (a BA rövidre záró elemet „M”-mel helyettesítve).

5.7. táblázat

	A	C	D	E	F
B	M	28	0(27)	27	32
C	22	C	0(3)	3	30
D	2	0(2)	D	3	31
E	3	0(0)	0(0)	E	0(30)
F	0(2)	27	28	0(3)	F

Látható, hogy pozitív AB döntés esetén az alsó határ értéke változatlan marad (206 km), mert minden sorban és oszlopban van nulla elem. Tehát a pozitív döntést választjuk, és ezen az ágon haladunk tovább.

A növekmény az EF relációban a legnagyobb. Az ~~EF~~ döntés választása esetén az alsó határ értéke 236 km lenne. Pozitív EF döntés esetén az 5.8. táblázatot kapjuk.

5.8. táblázat

	A	C	D	E
B	M	28	0(24)	24
C	22	C	0(0)	0(0)
D	2	0(27)	D	0(0)
F	0(29)	27	28	M

Az E oszlopában nem volt nulla elem, ezért redukálnunk kellett. Az alsó határ értéke ezért megnövekedett 206 + 3 = 209 km-re. A pozitív és a negatív döntést összehasonlítva egyértelmű, hogy a pozitív irányba megyünk tovább. Az új mátrixban az FA relációban legnagyobb a növekmény, negatív ~~FA~~ döntés esetén az alsó határ értéke 209 + 29 = 238 km lesz. Pozitív FA döntés esetén az 5.9. táblázatot kapjuk.

5.9. táblázat

	C	D	E
B	28	0(28)	M
C	C	0(0)	0(0)
D	0(28)	D	0(0)

Mivel minden sorban és oszlopban van nulla elem, az alsó határ értéke (209) nem változik. Tehát a pozitív döntést választjuk, és ezen az ágon megyünk tovább. Új mátrixunkban a növekmény a DC és a BD relációban azonos értékeket vesz fel. Az előzőekben már tárgyalt elvek alapján a DC relációt választjuk. Negatív ~~DC~~ döntés esetén az alsó határ értéke 209 + 28 = 237 km lesz. Pozitív DC döntés esetén az 5.10. táblázatot kapjuk, amelyet nem kell redukálni, ezért a körút hossza nem változik. Ezért a pozitív döntést választjuk.

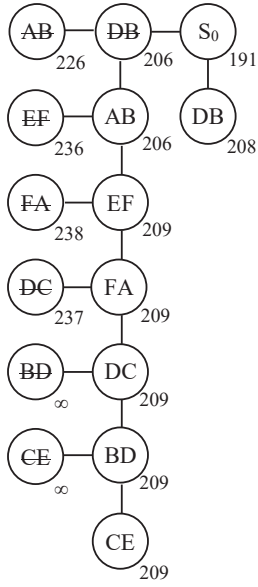
5.10. táblázat

	D	E
B	0(2M)	M
C	M	0(2M)

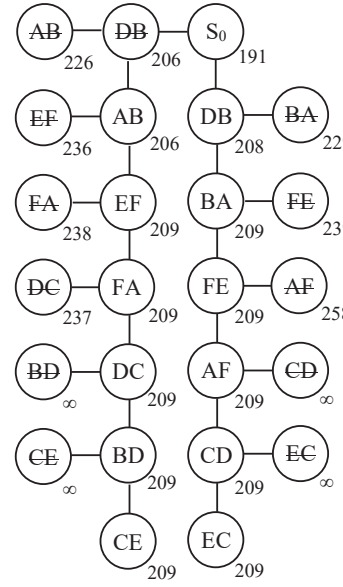
Látható, hogy az 5.10. táblázat a programozás utolsó lépését mutatja. A lehetséges BD és CE relációk közül bármelyiket is választjuk, a bejárás útvonala azonos marad. Bármelyik döntést választva a bejárási útvonalhossz alsó határának értéke változatlan marad, vagyis értéke 209 km lesz. A teljes döntési fát az 5.3 ábrán láthatjuk.

A pozitív döntéseink a következőek: AB, EF, FA, DC, BD, CE. Az A pontból kiindulva és sorba rendezve őket úgy, hogy a következő viszonylat kiinduló állomása az előző viszonylat érkező állomása legyen: AB, BD, DC, CE, EF, FA. A körút tehát: ABDCEF(A), hossza 209 km.

Lényeges megjegyezni azonban, hogy a DB ágon is tovább kell végezni a számításokat (5.4. ábra), mivel a DB pozitív döntés esetében a körút hosszának alsó határa (208) kisebb, mint az imént meghatározott körút hossza. A számításokat elvégezve azt kapjuk, hogy ezen ágon a körút hossza szintén 209 km lesz, iránya: AFECDBA. Ez lényegében ugyanaz a körút, csak ellentétes irányban, azaz ez egy alternatív optimum. Szimmetrikus kiindulási mátrix esetében ugyanis két körút, amelyek csak a körüljárás irányában különböznek, azonos hosszúságúak.



5.3. ábra



5.4. ábra

5.2. A Croes-módszer

Mint már említettük, a körutazási probléma megoldása a gyakorlatban közelítő eredményt adó módszerekkel történik. Ez azért is szükséges, mert mint az előző feladatban is láthattuk, igen hosszú volt az eljárás, mire megkaptuk az optimális megoldást. A sokféle közelítő módszer közül mi G. A. Croes 1958-ban publikált eljárásának[13] némileg módosított változatát ismertetjük.

Ezen eljárás előnyei a következők:

- egyaránt felhasználható szimmetrikus és aszimmetrikus feladatok megoldására,
- kézi és gépi számításokhoz egyaránt alkalmas,
- gyors és viszonylag jó közelítő eredményt ad,
- a számítás bármely közbeeső pontban megszakítható.

A módszer alkalmazását az előző feladatnál már felhasznált gyakorlati példa megoldásával mutatjuk be. A megoldás menete két részből áll. Az első rész a feladatnak egy közelítő megoldását adja, a második rész a közelítő megoldás finomításából áll.

Első lépésként az induló körút felállítását kell elvégeznünk. Ez úgy történik, hogy a kezdőpontból kiindulva mindig a legközelebbi pontba haladunk. Nézzük meg az induló körút összeállítását az 5.11. táblázatban.

5.11. táblázat

	A	B	C	D	E	F
A	A →	25	80	36	40	45
	↑	↓				
B	25	B →	60 →	8	35	48
	↑			↓		
C	80 ←	60 ←	C	47	50	85
			↑	↓		
D	36	8	47	D →	26	62
			↑		↓	
E	40	35	50	26	E →	34
			↑			↓
F	45	48	85 ←	62 ←	34 ←	F

[13] Croes, G. A. (1958): A Method for Solving Traveling-Salesman Problems. *Operations Research,* Vol. 6. No. 6. 791–812.

Az A állomás sorában található elemek az A helynek a többitől vett távolságát mutatják meg. Az A pontból jobbra, vízszintesen haladva kell a legkisebb költségelemig menni, majd onnan függőlegesen lefele a főátlóban található betűjelig. Példánkban az A sor második eleme a legkisebb, értéke 25. Ez az elem az A és a B pontokat köti össze. Folytatva az induló program szerkesztését, a B pontból kiindulva, ennek sorában a negyedik elem értéke a legkisebb, szám szerint 8. Tehát a B pontból kiindulva a legrövidebb úton a D pontot értük el. A D pont sorában a legrövidebb út megtételével az E pontot érhetjük el, tehát ez lesz a következő állomás. Az E pontból folytatva a programozást, annak sorában az F pont az a még meg nem látogatott pont, amelyikhez a legkisebb érték tartozik, ezért ez lesz az induló program következő eleme. Az F pont sorában a legkisebb értékű az E ponthoz tartozó elem. Ezt az állomást azonban már korábban érintettük. A feladat megfogalmazásánál feltételként szerepelt, hogy egy állomást a körút folyamán csak egyszer érinthetünk. Ezért a következő legkisebb elemet kell kiválasztanunk, amelyet eddig még nem vontunk be a programba. Példánkban ez az elem a C pont, távolsága 85. Mivel a körbejárás során már minden elemet érintettünk, vissza kell lépni a kiindulópontra (CA = 85). Ezzel az induló programunk elkészült. Valamennyi állomást végigjártuk úgy, hogy lehetőség szerint a legrövidebb legyen a körút hossza. Példánkban ez az alábbiak szerint alakul: A, B, D E, F, C, A. Ez megfelel az AB, BD, DE, EF, FC, CA viszonylatok egymás utáni bejárásának, amelyek hosszainak összege adja az induló körút hosszát:

$$25 + 8 + 26 + 34 + 85 + 80 = 258 \text{ km}.$$

Bár az induló megoldás elkészítése során a körbejárásnál arra törekedtünk, hogy mindig a legkisebb távolságokat vegyük figyelembe, és valamennyi megállóhelyet csak egyszer érintsük, ennek ellenére nem biztos, hogy megoldásunk az optimumot megközelítő eredményt jelenti. Ahhoz, hogy ezt meghatározhassuk, át kell rendezni az induló mátrixunkat oly módon, hogy a főátlóban az állomásokat jelentő betűk az induló körútnak megfelelően helyezkedjenek el. Az átrendezést két lépésben kell végrehajtani: oszlopok és sorok szerint. Az induló programból az oszlopok rendezése után az 5.12. táblázatot kapjuk.

5.12. táblázat

	A	B	D	E	F	C
A	A	25	36	40	45	80
B	25	B	8	35	48	60
C	80	60	47	50	85	C
D	36	8	D	26	62	47
E	40	35	26	E	34	50
F	45	48	62	34	F	85

Mint látható, az oszlopok átrendezésénél az eredeti mátrix harmadik eleme kerül az utolsó helyre, a negyedik, ötödik és hatodik oszlop eggyel előbbre csúszik. A sorok szerinti átrendezésnél az oszlopok átrendezésénél alkalmazott eljárással dolgozunk. Az átrendezés után az 5.13. táblázatot kapjuk.

5.13. táblázat

	A	B	D	E	F	C
A	A	25	36	40	45	80
B	25	B	8	35	48	60
D	36	8	D	26	62	47
E	40	35	26	E	34	50
F	45	48	62	34	F	85
C	80	60	47	50	85	C

Az oszlopok és sorok szerinti átrendezés után egy új mátrixot kaptunk, amelyre jellemző, hogy:

- a főátlójában szereplő betűjelek sorrendje megegyezik az állomások induló programban meghatározott sorrendjével, és
- a legkisebb költségértékek a főátlóbeli elemek mellett szerepelnek.

A továbbiakban annak megállapítása a feladatunk, hogy az új mátrix optimális-e, vagy még lehet rajta javítani. Ezt úgynevezett inverziós vizsgálat segítségével döntjük el.

Inverzión jelen esetben olyan módszert értünk, amelynek segítségével meg lehet keresni azokat a két vagy több állomásból álló útszakaszokat, amelyeket ha ellentétes irányban járjuk végig, úgy a körút hossza csökkenthető.

Tehát az a feladatunk, hogy egy tetszőleges körútról egy olyan körútra térjünk át, amelynek eredményeként a körút hossza csökken. Egy tetszőleges körútból úgy származtathatunk újabb körutakat, hogy annak egy-egy szakaszán a sorrendet megfordítjuk. A megfelelő szakasz kiválasztásához végre kell hajtani az inverziós vizsgálatot. Ez az alábbiak szerint történik.

Első lépésben bővítjük a mátrixot. A mátrix első sora fölé lemásoljuk az utolsó sor elemeit, az utolsó oszlop mellé pedig az első oszlop elemeit írjuk le. A betűk helyére 0-t írunk. Erre azért van szükség, hogy akár A-val kezdődő, akár F-fel végződő szakaszokat akarunk invertálni, az inverziót végre tudjuk hajtani (5.14. táblázat).

5.14. táblázat

80	60	47	50	85	0	
A	25	36	40	45	80	0
25	B	8	35	48	60	25
36	8	D	26	62	47	36
40	35	26	E	34	50	40
45	48	62	34	F	85	45
80	60	47	50	85	C	80

A következő lépés az inverziók képzése. A főátlóban elhelyezkedő pontok közül kettőt kiválasztunk. Erre a relációra olyan derékszögű háromszöget rajzolunk, amelynek átfogója az adott reláció végpontjait köti össze. Az AB relációt választva az 5.15. táblázat szemlélteti a háromszög elhelyezkedését. Ezt követően egyszerű számítást kell elvégeznünk. Az A pont fölött, tehát a háromszög felső csúcspontja fölött és a B csúcstól jobbra, tehát a háromszög alsó csúcspontja mellett található számokat összeadjuk. Példánkban az A fölötti érték 80, míg a B melletti 8. E két elemet félkövérrel jelöltük. A háromszög derékszöge fölött és mellett lévő számértékeket is összeadjuk, vagyis a 60-at és a 36-ot. Ezeket dőlt betűvel jelöltük. Az AB inverziót ezen két összeg különbségeként kapjuk:

$$L(AB) = (80 + 8) - (60 + 36) = -8$$

5.15. táblázat

80	*60*	47	50	85	0	
~~A~~	~~25~~	*36*	40	45	80	0
25	**B**	**8**	35	48	60	25
36	8	D	26	62	47	36
40	35	26	E	34	50	40
45	48	62	34	F	85	45
80	60	47	50	85	C	80

A fenti számítás csak szimmetrikus mátrixok esetén alkalmazható, aszimmetrikus mátrixnál némileg módosul az eljárás.

Az inverziókat valamennyi relációra fel kell írni annak érdekében, hogy ki tudjuk választani azt a relációt, amelyik a legnagyobb mértékben csökkenti a program értékét. Nézzük meg az 5.15. táblázat összes inverzióját és azok értékeit:

L(AB) = (80 + 8) – (60 + 36) = –8
L(AD) = (80 + 26) – (47 + 40) = 19
L(AE) = (80 + 34) – (50 + 45) = 19
L(AF) = (80 + 85) – (85 + 80) = 0
L(AC) = (80 + 80) – (0 + 0) = 160
L(BD) = (25 + 26) – (36 + 35) = –20
L(BE) = (25 + 34) – (40 + 48) = –29
L(BF) = (25 + 85) – (45 + 60) = 5
L(BC) = (25 + 80) – (80 + 25) = 0
L(DE) = (8 + 34) – (35 + 62) = –55
L(DF) = (8 + 85) – (48 + 47) = –2
L(DC) = (8 + 80) – (60 + 36) = –8
L(EF) = (26 + 85) – (62 + 50) = –1
L(EC) = (26 + 80) – (47 + 40) = 19
L(FC) = (34 + 80) – (50 + 45) = 19

Látható, hogy az inverziós számítások eredményeként pozitív, negatív vagy nulla értékeket kaphatunk. Azonban csak azoknak a relációknak a felcserélése javítja a programot, amelyekre pozitív értékeket kaptunk. Nulla inverziójú viszonylat megfordítása esetén a körút hossza nem változik, csak a körbejárás sorrendje módosul.

A pozitív eredményt adó inverziók közül azt a relációt cseréljük fel a programban, ahol az inverzió a legnagyobb pozitív érték. Több ilyen érték esetén tetszőleges a választás. Példánkban a legnagyobb pozitív érték 160. Ez azonban az AC viszonylathoz tartozó inverzió, amelynek megfordítása valójában a teljes körút megfordítását jelentené, azaz a hosszát nem változtatná meg, csak a bejárási sorrend lenne pont ellentétes. Ezt az inverziót tehát a számítások során sosem vesszük figyelembe.

A következő legnagyobb érték, 19, négy relációnál is megtalálható: AD, AE, EC, FC. A választás tetszőleges. Válasszuk az AE relációt. A reláció felcserélésével a közöttük levő állomások iránya megfordul, vagyis a kiinduló ABDEFC(A) körút ABDE útvonalrésze helyett EDBA-t írunk

a körútba: EDBAFC. Mivel a körút ciklikusan permutálható, az „utolsó előre fuss" elven úgy rendezzük, hogy az A állomás legyen a kiindulópont: CEDBAF, FCEDBA, AFCEDB(A). A fenti viszonylat felcserélésével kapott új körút hossza:

AF, FC, CE, ED, DB, BA

45 + 85 + 50 + 26 + 8 + 25 = 239 km.

Ha megvizsgáljuk az új körút hosszát, láthatjuk, hogy az az induló körúthoz viszonyítva (258 km) pontosan a kiválasztott AE reláció inverziójának értékével, 19-cel csökkent.

Azonban nem biztos, hogy az új körút kvázioptimális. Újból el kell végezni az inverziós vizsgálatot az új körútnak megfelelően módosított mátrix alapján (5.16. táblázat).

5.16. táblázat

25	48	60	35	8	0	
A	45	80	40	36	25	0
45	F	85	34	62	48	45
80	85	C	50	47	60	80
40	34	50	E	26	35	40
36	62	47	26	D	8	36
25	48	60	35	8	B	25

Ha az inverziók között ismét lesz pozitív, akkor a program javítását az ismertetett módszerrel folytatni kell. Mindaddig javítjuk a programot, amíg sehol sem kapunk pozitív eredményű inverziót. Ebben az esetben kvázioptimális eredményhez jutottunk. Az 5.16. táblázathoz tartozó inverziók:

L(AF) = (25 + 85) – (48 + 80) = –18
L(AC) = (25 + 50) – (60 + 40) = –25
L(AE) = (25 + 26) – (36 + 36) = –20
L(AD) = (25 + 8) – (8 + 25) = 0
L(AB) = (25 + 25) – (0 + 0) = 50
L(FC) = (45 + 40) – (80 + 34) = –29
L(FE) = (45 + 26) – (40 + 62) = –31
L(FD) = (45 + 8) – (36 + 48) = –31
L(FB) = (45 + 25) – (25 + 45) = 0
L(CE) = (85 + 26) – (34 + 47) = 30
L(CD) = (85 + 8) – (62 + 60) = –29

L(CB) = (85 + 25) – (48 + 80) = –18
L(ED) = (50 + 8) – (47 + 35) = –24
L(EB) = (50 + 25) – (60 + 40) = –25
L(DB) = (26 + 25) – (35 + 36) = –20

Az AB szakasz invertálása csak a körút irányát fordítja meg, de a CE szakasz invertálása valós pozitív eredményt ad. Ezen viszonylat felcserélésével az új körút a következő: AFECDB(A), amelynek hossza az egyes elemi viszonylatok összegeként adódik:

AF, FE, EC, CD, DB, BA
45 + 34 + 50 + 47 + 8 + 25 = 209 km.

Még most sem biztos, hogy elérkeztünk a jó megoldáshoz. Ismételten el kell végezni az inverziós vizsgálatot az új körútnak megfelelően módosított mátrix segítségével (5.17. táblázat).

5.17. táblázat

25	48	35	60	8	0	
A	45	40	80	36	25	0
45	F	34	85	62	48	45
40	35	E	50	26	35	40
80	85	50	C	47	60	80
36	62	26	47	D	8	36
25	48	35	60	8	B	25

Az inverziók értékei:

L(AF) = –29
L(AE) = –40
L(AC) = –24
L(AD) = 0
L(AB) = 50
L(FE) = –30
L(FC) = –50
L(FD) = –31
L(FB) = 0
L(EC) = –61
L(ED) = –55
L(EB) = –29

L(CD) = –28
L(CB) = –40
L(DB) = –24

Mint láthatjuk, az AB viszonylat, a körút teljes megfordítása kivételével sehol sem kaptunk pozitív eredményt. További javításra tehát nincs szükség, elértük a bemutatott módszerrel elérhető legjobb eredményt. Kisebb feladatoknál gyakran az optimális körutat is megkaphatjuk, nagyobb feladatoknál (50 körüli állomásszámnál) kb. 4-5%-os pontosságú eredményt ad a módszer.

5.3. Megoldás magyar módszerrel

A körutazási probléma azonban felfogható egy speciális hozzárendelési feladatként is: minden város „kibocsátóhely" is és „fogadóhely" is, de csak pontosan egy másik város, mint „fogadóhely", illetve „kibocsátóhely" számára. Ez a klasszikus hozzárendelési feladat jelölésével megfogalmazva azt jelenti, hogy a kibocsátóhelyek megegyeznek a fogadóhelyekkel: $K_i = F_i$. Így tekintve a feladatot az megoldható a magyar módszerrel. Nézzük ezt meg egy példán keresztül.

Legyen adott a költségmátrix az 5.18. táblázat szerinti kilométerértékekkel.

5.18. táblázat

	A	B	C	D	E	
A	A	12	8	17	9	1
B	13	B	8	1	13	1
C	3	1	C	19	15	1
D	17	2	1	D	2	1
E	13	13	6	9	E	1
	1	1	1	1	1	

Redukáljuk a mátrixunkat sorok, majd oszlopok szerint (5.19. és 5.20. táblázat, az azonosan 1 kapacitást, illetve igényt elhagytuk):

5.19. táblázat

	A	B	C	D	E	u_i
A	A	4	0	9	1	8
B	12	B	7	0	12	1
C	2	0	C	18	4	1
D	16	1	0	D	1	1
E	7	7	0	3	E	6
						$\sum u_i = 17$

5.20. táblázat

	A	B	C	D	E
A	A	4	0	9	0
B	10	B	7	0	11
C	0	0	C	18	13
D	14	1	0	D	0
E	5	7	0	3	E
v_j	2	0	0	0	1

$\sum v_j = 3$

A 3.1. alfejezetben tanult módon programozva a második és az ötödik sorban van szabad nulla (c_{24} és c_{53}), majd oszloponként programozva az első és az ötödik oszlopban van szabad nulla (c_{31} és c_{15}). Az induló program az 5.21. táblázatban látható.

5.21. táblázat

	A	B	C	D	E
A	A	4	~~0~~	9	[0]
B	10	B	7	[0]	11
C	[0]	~~0~~	C	18	13
D	14	1	~~0~~	D	~~0~~
E	5	7	[0]	3	E

Mivel minden szabad nullára programoztunk, de csak négy kötött elemünk van öt helyett, javítanunk kell az induló programunkat. A negyedik sort és a második oszlopot nem vontuk be a programba, ezért ezen sor és oszlop

nulla elemeire merőlegesen húzunk fedővonalakat, azaz a harmadik és az ötödik oszlopot, valamint a harmadik sort fedjük le. Ekkor azonban a c_{24} nulla elemet még nem fedtük le. Az erre húzott fedővonal lehet függőleges is, vízszintes is, mivel egyik módon sem metszi két fedővonal kötött elemen egymást. Válasszuk most a vízszintest, azaz fedjük le a második sort. Az így kapott fedővonalrendszer az 5.22. táblázatban látható.

5.22. táblázat

	A	B	C	D	E
A	A	4	0	9	0
B	10	B	7	0	11
C	0	0	C	18	13
D	14	1	0	D	0
E	5	7	0	3	E

A legkisebb le nem fedett elem a c_{42}, azaz $\varepsilon_1 = 1$. Javítsuk a mátrixunkat a 3.2. alfejezetben megismert módon az ε-transzformációval. A javított mátrix az 5.23. táblázatban látható.

5.23. táblázat

	A	B	C	D	E
A	A	3	0	8	0
B	10	B	8	0	12
C	0	0	C	18	14
D	13	0	0	D	0
E	4	6	0	2	E

Programozzunk ismét soronként, illetve oszloponként: a c_{24} és a c_{53}, majd a c_{31}, c_{44} és c_{15} elemekre való programozás után öt kötött elemet kapunk (5.24. táblázat).

5.24. táblázat

	A	B	C	D	E
A	A	3	0	8	0
B	10	B	8	0	12
C	0	0	C	18	14
D	13	0	0	D	0
E	4	6	0	2	E

Azt gondolhatnánk, hogy ezzel meg is oldottuk a feladatot. Ha ez valóban egy hozzárendelési feladat lenne, ez igaz is lenne. Azonban itt egy körutazási problémával van dolgunk, ahol van egy eddig ki nem mondott plusz feltételünk is: minden várost meg kell látogatni. Vizsgáljuk meg tehát a kapott körutat, hogy megfelel-e ennek a feltételnek. Az A városból kiindulva a körutunk: A–E–C–A, azaz a B és a D városokat nem érintve érkeztünk vissza a kiinduló városba. Valójában tehát két kisebb, független körutat kaptunk (a másik körút a B–D–B út).

Mivel öt-öt sorunk és oszlopunk és öt kötött elemünk van, a Kőnig–Egerváry-tétel alapján a táblázatunkat javítani az ismert módon nem tudjuk. A probléma feloldása az, ami a hozzárendelési problémánál nem volt megengedett: nem csak nulla elemekre fogunk programozni. Bevonjuk a következő legkisebb elemet, jelen esetben a 2-est, és minden tekintetben úgy kezeljük, mintha 0 lenne. Először erre a 2-es számra, a c_{54} elemre programozunk, hogy biztosan kötött elem legyen, és nehogy egy 0-ra programozásnál áthúzzuk, ezzel egyidejűleg pedig a c_{24} és a c_{53} elemeket áthúzzuk, mivel az ötödik sort és a negyedik oszlopot ezzel már elprogramoztuk.

Ezután a szokott módon programozunk soronként és oszloponként a c_{31}, a c_{42} és a c_{13} elemekre (5.25. táblázat). Ha lenne még 2-es érték a táblázatban, arra is programozhatnánk, illetve ugyanúgy át kellene húznunk, mintha nulla lenne, de ez itt most nem áll fenn.

5.25. táblázat

	A	B	C	D	E
A	A	3	[0]	8	~~0~~
B	10	B	8	~~0~~	12
C	[0]	~~0~~	C	18	14
D	13	[0]	~~0~~	D	~~0~~
E	4	6	~~0~~	[2]	E

Így azonban megint csak négy kötött elemet kaptunk, ezért bevonjuk a következő legkisebb elemet is, a 3-at. Először erre, a c_{13} elemre programozunk, és a merőleges sorban és oszlopban szereplő, legfeljebb 3 értékű elemeket (c_{13}, c_{15}, c_{32}, c_{42}) áthúzzuk. Ezután elvégezzük a programozást: c_{24} (c_{54}-et áthúzzuk, hiszen a legfeljebb három értékű elemeket is úgy kezeljük, mint a nullákat), c_{31}, c_{53} (c_{43}-at áthúzzuk), c_{45}. A programozás eredménye az 5.26. táblázatban látható.

5.26. táblázat

	A	B	C	D	E
A	A	[3]	~~0~~	8	~~0~~
B	10	B	8	[0]	12
C	[0]	~~0~~	C	18	14
D	13	~~0~~	~~0~~	D	[0]
E	4	6	[0]	~~2~~	E

Kaptunk öt kötött elemet, ezért leellenőrizzük, hogy az A városból kiinduló út valóban érinti-e az összes többi várost: A–B–D–E–C–A. Most minden várost körbejárunk, tehát megtaláltuk az optimális körutat. Határozzuk még meg a hosszát az 5.18. táblázat távolságértékei alapján: 12 + 1 + 2 + 6 + 3 = 24 km.

6. Legrövidebbút-keresés

Gyakran előfordul, hogy nem több kibocsátóhelyről több fogadóhelyre történő szállítást kell megtervezni, hanem két pont között kell a legrövidebb utat megkeresni. A „legrövidebb" itt nem csak távolságot jelenthet, annál bővebb jelentést hordoz: lehet szó időben leggyorsabb, legkisebb költségű, legkevesebb szintvonalátlépést vagy legkevesebb repülés közbeni leszállást igénylő útvonalról.

A hálózatot, amelyben két adott pont között keressük az (ebben a tágabb értelemben vett) legrövidebb utat, általában egy gráffal szoktuk modellezni. Egy gráf csúcsok és az ezeket összekötő élek halmaza. A csúcsok lehetnek városok, megállók, repülőterek vagy egyszerűen földrajzi pontok, az élek pedig az ezeket összekötő lehetséges utak. Egy város utcahálózatát lehet például egy olyan gráffal jellemezni, amelyben az egyes kereszteződések a csúcspontok, a köztük levő utcák pedig az élek. Ezek az élek lehetnek irányítatlanok, amikor mindkét irányban lehet az utcában közlekedni, vagy irányítottak, egyirányú utcák esetében, amikor az egyik irányból a forgalom tiltott.

Az egyes csomópontok közötti utcaszakaszok azonban (ennél a példánál maradva) eltérő hosszúságúak. Ezért minden élhez hozzárendelünk egy úgynevezett súlyt, amely jelen esetben az utca hosszát jelenti. Ez a súly természetesen a megoldandó problémától függően lehet menetidő, üzemanyag-fogyasztás, szintemelkedés vagy akár egy komfortot jellemző számérték is. Az ilyen, súlyozott éleket (is) tartalmazó gráfot súlyozott gráfnak nevezzük.

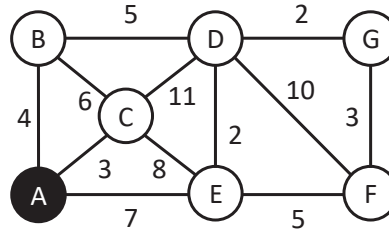
A legrövidebbút-keresés célja megtalálni két pont között azt az utat, amelyik esetében a bejárt élekhez tartozó súlyok összege minimális.

Ennek a problémának a megoldására két módszert mutatunk be, amelyek a legelterjedtebbek a gyakorlati alkalmazás területén. Az első módszer csak pozitív súlyú éleket tartalmazó gráfra, a második negatív éleket tartalmazóra (mivel például regeneratív fékezésű jármű lejtőn lefele gurulva a hálózatba visszatáplálható áramot tud termelni, ezért ezen a szakaszon a fogyasztása negatív) is alkalmazható.

6.1. A Dijkstra-algoritmus

Adott egy csak pozitív súlyú éleket tartalmazó gráf. Keressük meg benne egy adott kiindulási és egy adott érkezési pont között a legrövidebb utat. A problémát 1956-ban oldotta meg Edsger Wybe Dijkstra (1930–2002) holland matematikus, de csak 1959-ben publikálta azt.[14] Ilyen típusú gráfokban ma is ennek, illetve az eredeti algoritmus egy módosított változatának a használata a legelterjedtebb, amely a kiindulási csúcs és az összes többi csúcs közötti legrövidebb utat megadja. A módszert egy konkrét példán mutatjuk be.

Adott tehát a 6.1. ábrán látható gráf, amelyben keressük meg az A csúcs és a többi csúcs közötti legrövidebb utakat.



6.1. ábra

Ennek meghatározásához egy táblázatot vezetünk az egyes csúcsok A-tól való távolságáról, amely kezdetben csak A-ra tartalmaz 0 értéket (mivel ott vagyunk), minden másik csúcsra végtelen a távolság, hiszen még nem kezdtük el a számolást (6.1. táblázat). Az algoritmus során végiglátogatjuk a gráf összes csúcsát A-tól indulva úgy, hogy mindig a legközelebbi csúcsba megyünk, és minden csúcsot csak egyszer látogatunk meg. Minden lépésben meghatározzuk, hogy az adott csúcs szomszédai milyen hosszúságú úton érhetőek el, és ha ez az érték kisebb, mint a csúcsokra a táblázatban szereplő érték, akkor frissítjük a táblázatot.

[14] Dijkstra, Edsger Wybe (1959): A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik,* Vol. 1. No. 1. 269–271.

Induljunk el tehát az A csúcsból. Innen a B, C, E csúcsok érhetőek el, távolságuk rendre 4, 3 és 7. Ezek mind kisebbek, mint végtelen, frissítsük tehát a táblázatunkat. A D, F, G csúcsok nem érhetőek el az A csúcsból közvetlenül, vagyis tőle vett távolságuk végtelen. Ez nem kisebb a táblázatban szereplő értéknél, ezért ezeket változatlanul hagyjuk. Az első lépés frissítései után a 6.2. táblázatot kapjuk. Az A csúcsot megjelöljük, hogy már meglátogattuk.

6.1. táblázat

A	0
B	∞
C	∞
D	∞
E	∞
F	∞
G	∞

6.2. táblázat

A ✓	0
B	4
C	3
D	∞
E	7
F	∞
G	∞

A még meg nem látogatott csúcsok közül a legkisebb távolságra a C csúcs van, ide megyünk a következő lépésben. A C csúcsból elérhetőek az A, B, D, E csúcsok, távolságuk rendre 3, 6, 11, 8. Ehhez az értékhez azonban hozzá kell adni az AC távolságot (3), ugyanis ezen az úton jutottunk el ide, ahonnan továbbmegyünk. Így az elérhető csúcsok távolságai 6, 9, 14, 11. Mivel az A, a B és az E csúcsra $6 > 0$, $9 > 4$ és $8 > 7$, azaz ezek a csúcsok a C csúcson keresztül hosszabb úton érhetőek el, mint közvetlenül az A csúcsból, ezért a sorukat nem frissítjük a táblázatban. A D csúcsra azonban $14 < \infty$, így itt az új értéket írjuk be. Az F és a G csúcs továbbra is elérhetetlen, távolságuk marad ∞. A C csúcsot megjelöljük, hogy már meglátogattuk (6.3. táblázat).

Az A-hoz legközelebbi meg nem látogatott csúcs a B csúcs. Innen az A, C, D csúcsok érhetőek el, távolságuk rendre 4, 6, 5. Ehhez hozzáadva a B-be jutáshoz szükséges utat (4), kapjuk: 8, 10, 9. Mivel az A és a C csúcsra $8 > 0$ és $6 > 3$, ezeket az értékeket nem frissítjük. A D csúcsra azonban $9 < 14$, így a kisebb értéket írjuk a táblázatba. Az E, F és a G csúcsok elérhetetlenek a B-ből, távolságuk változatlan marad. A B csúcsot megjelöljük, hogy már meglátogattuk (6.4. táblázat).

6.3. táblázat

A ✓	0
B	4
C ✓	3
D	14
E	7
F	∞
G	∞

6.4. táblázat

A ✓	0
B ✓	4
C ✓	3
D	9
E	7
F	∞
G	∞

Az A-hoz legközelebbi meg nem látogatott csúcs az E csúcs. Innen az A, C, D, F csúcsok érhetők el, távolságaik rendre 7, 8, 2, 5. Ezekhez hozzáadva az E csúcsba jutáshoz szükséges utat (7), az egyes csúcsokra kapjuk: A 14 > 0, C 15 > 3, D 17 > 9, F 12 < ∞, azaz csak az F csúcs értékét kell frissíteni. A B és a G csúcsok elérhetetlenek az E-ből, távolságuk változatlan marad. Az E csúcsot megjelöljük, hogy már meglátogattuk (6.5. táblázat).

Az A-hoz legközelebbi meg nem látogatott csúcs a D csúcs. Innen a B, C, E, F, G csúcsok érhetőek el, távolságuk rendre 5, 11, 2, 10 és 2. Ezekkel az értékekkel megnövelve a D csúcs A-tól való távolságát (9), kapjuk: 14, 20, 11, 19, 11. Ezek közül csak az utolsó, az F csúcsra vonatkozó érték kisebb a táblázatban szereplő értéknél, ezért csak azt frissítjük (az A csúcs el sem érhető D-ből, ezért az is változatlan marad). A D csúcsot megjelöljük, hogy már meglátogattuk (6.6. táblázat).

6.5. táblázat

A ✓	0
B ✓	4
C ✓	3
D	9
E ✓	7
F	12
G	∞

6.6. táblázat

A ✓	0
B ✓	4
C ✓	3
D ✓	9
E ✓	7
F	12
G	11

Az A-hoz legközelebbi, meg nem látogatott csúcs a G csúcs. Innen a D és az F csúcsok érhetők el, távolságuk rendre 2 és 3. Az AG távolsághoz (11) hozzáadva ezeket az értékeket 13-at és 14-et kapunk, amelyek nagyobbak a D és az F csúcsra a táblázatban szereplő értéknél, ezért az nem változik.

Hasonlóan az utolsó, az F csúcsot meglátogatva sem csökken egyik érték sem. Ezzel az algoritmus végére értünk. A 6.7. táblázatban egymás mellett láthatjuk a 6.2.–6.6. táblázatokban szereplő értékeket. Innen az egyes útvonalakat tudjuk meghatározni: amelyik lépésben a végső, legkisebb értékére csökkent az adott csúcs távolsága, oda abból a csúcsból jutottunk, amelyik az oszlop fejlécében szerepel.

6.7. táblázat

	A	C	B	E	D
A	0	0	0	0	0
B	4	4	4	4	4
C	3	3	3	3	3
D	∞	14	9	9	9
E	7	7	7	7	7
F	∞	∞	∞	12	12
G	∞	∞	∞	∞	11

Ez a következőt jelenti. Az A csúcs önmagától vett távolsága nulla.

Az első oszlopban, vagyis az A csúcsból éri el minimális értékét a B, a C és az E csúcs távolsága, vagyis ezek közvetlenül az A csúcsból érhetők el a legrövidebb úton: AB, AC, AE.

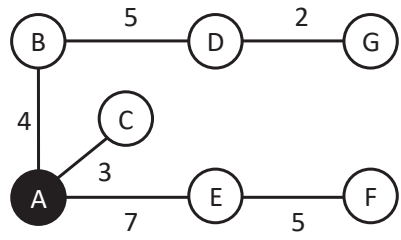
A D csúcs A csúcstól vett távolsága a harmadik, azaz a B csúcshoz tartozó oszlopban éri el minimumát: BD. De a B csúcsba is el kellett valahogy jutni. Ez a legrövidebb úton közvetlenül az A csúcsból lehetséges, ahogy az imént láttuk: AB+BD → ABD.

Az F és a G csúcs távolsága a negyedik, illetve az ötödik oszlopban lesz minimális, vagyis ezek az E és a D csúcsból érhetők el a legrövidebb úton, vagyis az EF és a DG út része az A csúcs és ezen csúcsok közti legrövidebb útnak. De még az eddig meghatározott legrövidebb utak alapján ezeket is vissza kell vezetni az A csúcsba: AE+EF → AEF és ABD+DG → ADBG.

Összefoglalva tehát:

A: 0 AB: 4 AC: 3 ABD: 9 AE: 7 AEF: 12 ABDG: 11

A legrövidebb utakat a 6.2. ábra mutatja.



6.2. ábra

A fentebb elmondottak irányított gráfokra ugyanúgy érvényesek, egyszerűen némely csúcsból kevesebb másik csúcs lesz elérhető, de az elv azonos.

6.2. A Bellman–Ford-algoritmus

Lehetséges azonban negatív súlyú éleket is tartalmazó gráf. Ezekre a Dijkstra-algoritmus nem használható, megoldásukra 1956-ban dolgozták ki Lester Randolph Ford, Jr. (1927–2017),[15] tőle függetlenül 1958-ban Richard Ernest Bellman (1920–1985)[16] és mindkettőjüktől függetlenül 1959-ben Edward Forrest Moore (1925–2003)[17] amerikai matematikusok a végül az első két felfedezőjéről elnevezett Bellman–Ford-algoritmust.

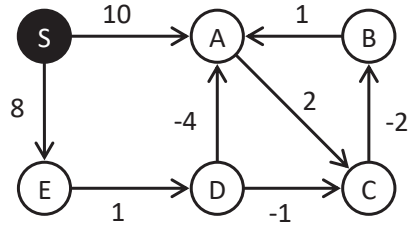
Ennél a módszernél csak az a kitétel, hogy negatív súlyú kör ne legyen a gráfban, azaz ne létezzen benne olyan zárt út, amelyen körbemenve az élek összsúlya negatív lenne. Ekkor ugyanis egyszer eljutva ezen körút egyik csúcsába bármely út értékét tetszőlegesen nagy negatív számmá tehetnénk.

Legyen adott a 6.3. ábrán látható irányított súlyozott gráf. Keressük meg a legrövidebb utakat S-ből a többi csúcsba.

[15] Ford, Lester R. Jr. (1956): *Network Flow Theory.* Santa Monica, California, Rand Corporation.

[16] Bellman, Richard (1958): On a Routing Problem. *Quarterly of Applied Mathematics,* Vol. 16. No. 1. 87–90.

[17] Moore, Edward F. (1959): The shortest path through a maze. *Proc. Internat. Sympos. Switching Theory 1957, Part II.* Cambridge, Mass., Harvard Univ. Press. 285–292.



6.3. ábra

A Bellman–Ford-algoritmus egy iteratív algoritmus, azaz lépésenként közelítve, iterálva jutunk el a megoldáshoz. Az iterációk száma legfeljebb a csúcsok száma mínusz egy. Előbb is eljuthatunk az optimumig, de több iterációra biztosan nincs szükség. Esetünkben tehát legfeljebb öt iterációra lesz szükség.

Kezdetben minden csúcs S-től vett távolsága végtelen, csak S önmagától vett távolsága nulla (6.8. táblázat).

6.8. táblázat

S	0
A	∞
B	∞
C	∞
D	∞
E	∞

Első iteráció

Végignézzük, hogy az S csúcsból a többi csúcs milyen hosszúságú úton érhető el. Ha ez az érték kisebb, mint a táblázatban szereplő, frissítjük a táblázatot. Ezután sorban végignézzük az A, B, C, D és E csúcsokra is, hogy a többi csúcs milyen hosszúságú úton érhető el: hozzáadjuk az útszakasz hosszát a vizsgált kiindulócsúcs táblázatban szereplő értékéhez, és ha ez kisebb, mint ami a táblázatban szerepel, frissítjük az értéket.

Az S csúcsból az A és az E csúcs érhető el végtelennél rövidebb úton (azaz ezek érhetőek el közvetlenül), 10, illetve 8 egységnyi hosszúságú úton.

Az A csúcsból csak a C csúcs érhető el, az AC távolság 2. Ezt hozzáadva az SA távolság aktuális értékéhez, 10-hez, az SC távolságra 12-t kapunk, ezzel frissítjük a táblázatot, mivel ez kisebb érték, mint a jelenleg a táblázatban szereplő SA távolság.

A B csúcs S-től vett távolsága jelenleg végtelen, ezért ezt kihagyjuk.

A C csúcsból csak a B csúcs érhető el, a CB távolság –2. Ezt hozzáadva az SC távolság aktuális értékéhez, 12-höz, az SB távolságra 10-et kapunk, ezzel frissítjük a táblázatot.

A D csúcs S-től vett távolsága jelenleg végtelen, ezért ezt is kihagyjuk most.

Az E csúcsból csak a D csúcs érhető el, az ED távolság 1. Ezt hozzáadva az SE távolság aktuális értékéhez, 8-hoz, az SD távolságra 9-et kapunk, ezzel frissítjük a táblázatot.

Ezzel vége az első iterációnak, a kapott eredmények a 6.9. táblázatban láthatóak.

6.9. táblázat

	S-ből	A-ból	B-ből	C-ből	D-ből	E-ből
S	0	0	∞, kihagyjuk	0	∞, kihagyjuk	0
A	10	10		10		10
B	∞	∞		10		10
C	∞	12		12		12
D	∞	∞		∞		9
E	8	8		8		8

Második iteráció

Az előző táblázat utolsó oszlopából indulunk ki, és ismét végignézzük az összes csúcsra, hogy mi érhető el onnan, és milyen hosszú úton.

Az S csúcsból az A és az E csúcs érhető el 10, illetve 8 egységnyi hosszúságú úton, ezek az értékek megegyeznek a táblázatban szereplő értékkel, ezért nem frissítjük azokat (az SS távolság 0).

Az A csúcsból csak a C csúcs érhető el, az AC távolság 2. Ezt hozzáadva az SA távolság aktuális értékéhez, 10-hez, az SC távolságra 12-t kapunk, ez azonos a táblázatban szereplő értékkel, ezért nem frissítjük a táblázatot.

A B csúcsból csak az A csúcs érhető el, a BA távolság 1. Ezt hozzáadva az SB távolság aktuális értékéhez, 10-hez, az SA távolságra 11-et kapunk, de ez nagyobb, mint a táblázatban található érték, ezért nem frissítjük azt.

A C csúcsból csak a B csúcs érhető el, a CB távolság –2. Ezt hozzáadva az SC távolság aktuális értékéhez, 12-höz, az SB távolságra 10-et kapunk, ez azonos a táblázatban szereplő értékkel, ezért nem frissítjük a táblázatot.

A D csúcsból az A és a C csúcsok érhetők el, a DA távolság –4, a DC távolság –1. Ezeket hozzáadva az SD távolság aktuális értékéhez, 9-hez, az SA távolságra 5-öt, az SC távolságra 8-at kapunk. Mivel mindkettő kisebb a táblázatban szereplő értéknél, ezért frissítjük azt.

Az E csúcsból csak a D csúcs érhető el, az ED távolság 1. Ezt hozzáadva az SE távolság aktuális értékéhez, 8-hoz, az SD távolságra 9-et kapunk, ez azonos a táblázatban szereplő értékkel, ezért nem frissítjük a táblázatot.

Ezzel vége a második iterációnak, a kapott eredmények a 6.10. táblázatban láthatóak.

6.10. táblázat

	S-ből	A-ból	B-ből	C-ből	D-ből	E-ből
S	0	0	0	0	0	0
A	10	10	10	10	5	5
B	10	10	10	10	10	10
C	12	12	12	12	8	8
D	9	9	9	9	9	9
E	8	8	8	8	8	8

Harmadik iteráció

Ismét az előző, a 6.10. táblázat utolsó oszlopából indulunk ki, és ismét egyesével megvizsgáljuk a csúcsokat.

Az S csúcsból az A és az E csúcs érhető el 10, illetve 8 egységnyi hosszúságú úton, ezek az értékek megegyeznek a táblázatban szereplő értékkel.

Az A csúcsból csak a C csúcs érhető el, az AC távolság 2. Ezt hozzáadva az SA távolság aktuális értékéhez, 5-höz, az SC távolságra 7-et kapunk, ami kisebb a táblázatban szereplő értéknél, ezért frissítjük a táblázatot.

A B csúcsból csak az A csúcs érhető el, a BA távolság 1. Ezt hozzáadva az SB távolság aktuális értékéhez, 10-hez, az SA távolságra 11-et kapunk, de ez nagyobb, mint a táblázatban található érték, ezért nem frissítjük azt.

A C csúcsból csak a B csúcs érhető el, a CB távolság –2. Ezt hozzáadva az SC távolság aktuális értékéhez, 7-hez, az SB távolságra 5-öt kapunk, ami kisebb a táblázatban szereplő értéknél, ezért frissítjük a táblázatot.

A D csúcsból az A és a C csúcsok érhetők el, a DA távolság –4, a DC távolság –1. Ezeket hozzáadva az SD távolság aktuális értékéhez, 9-hez, az SA távolságra 5-öt, az SC távolságra 8-at kapunk. Az SC-re kapott távolság nagyobb a táblázatban szereplő értéknél, az SA-ra kapott pedig azonos vele, ezért egyiket sem frissítjük.

Az E csúcsból csak a D csúcs érhető el, az ED távolság 1. Ezt hozzáadva az SE távolság aktuális értékéhez, 8-hoz, az SD távolságra 9-et kapunk, ez azonos a táblázatban szereplő értékkel, ezért nem frissítjük a táblázatot.

Ezzel vége a harmadik iterációnak, a kapott eredmények a 6.11. táblázatban láthatóak.

6.11. táblázat

	S-ből	A-ból	B-ből	C-ből	D-ből	E-ből
S	0	0	0	0	0	0
A	5	5	5	5	5	5
B	10	10	10	5	5	5
C	8	7	7	7	7	7
D	9	9	9	9	9	9
E	8	8	8	8	8	8

Negyedik iteráció

Kiindulva a 6.11. táblázat utolsó oszlopából ismét megnézzük, hogy az egyes csúcsokból hova mekkora úton lehet eljutni.

Az S csúcsból az A és az E csúcs érhető el 10, illetve 8 egységnyi hosszúságú úton, ezek az értékek megegyeznek a táblázatban szereplő értékkel.

Az A csúcsból csak a C csúcs érhető el, az AC távolság 2. Ezt hozzáadva az SA távolság aktuális értékéhez, 5-höz, az SC távolságra 7-et kapunk, ami azonos a táblázatban szereplő értékkel.

A B csúcsból csak az A csúcs érhető el, a BA távolság 1. Ezt hozzáadva az SB távolság aktuális értékéhez, 5-höz, az SA távolságra 6-ot kapunk, de ez nagyobb, mint a táblázatban található érték.

A C csúcsból csak a B csúcs érhető el, a CB távolság –2. Ezt hozzáadva az SC távolság aktuális értékéhez, 7-hez, az SB távolságra 5-öt kapunk, ami azonos a táblázatban szereplő értékkel.

A D csúcsból az A és a C csúcsok érhetőek el, a DA távolság –4, a DC távolság –1. Ezeket hozzáadva az SD távolság aktuális értékéhez, 9-hez, az SA távolságra 5-öt, az SC távolságra 8-at kapunk. Az SC-re kapott távolság nagyobb a táblázatban szereplő értéknél, az SA-ra kapott pedig azonos vele.

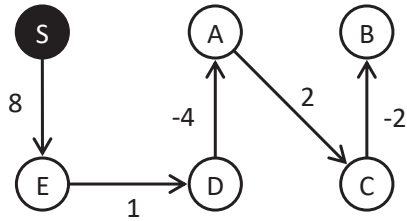
Az E csúcsból csak a D csúcs érhető el, az ED távolság 1. Ezt hozzáadva az SE távolság aktuális értékéhez, 8-hoz, az SD távolságra 9-et kapunk, ez azonos a táblázatban szereplő értékkel.

Ezzel vége a negyedik iterációnak. Az iteráció során a táblázatban szereplő értékek nem változtak, ezért az algoritmus végére értünk. Jelen esetben nem volt már szükség az ötödik iteráció elvégzésére, de ha szükség lett volna rá, mert a negyedik iteráció során változtak volna még a táblázatban szereplő értékek, sőt, még ha az ötödik iteráció során is változtak volna, az ötödik iteráció végére akkor is biztosan az optimális megoldást kaptuk volna meg.

Végignézve az egyes iterációk táblázatait visszafelé haladva és megkeresve, hogy az egyes csúcsok útjai melyik csúcsból kiindulva érték el a minimális értéküket, meghatározhatjuk a konkrét útvonalakat. Csak egy konkrét példára, az A csúcsra megnézve: a távolsága az S csúcstól 5. Ezt a második iteráció során a D csúcsból kiindulva érte el. Az AD útvonalrészlet tehát biztos. A D csúcs távolsága az első iteráció során az E csúcsból kiindulva lett minimális, azaz az ED útvonalrészlet, így az EDA út is biztosan része az SA útnak. Az E csúcs távolsága az S csúcstól az első iteráció során az S csúcsból kiindulva lett minimális. Vagyis az SE út is biztosan része a legrövidebb SA útnak, de az SE és az EDA útrészleteket összeillesztve meg is kapjuk a pontos SA utat: SEDA.

Ezt minden csúcsra el kell végezni. A legrövidebb utak a következőképpen alakulnak (grafikusan a 6.4. ábra mutatja őket):

SEDA	5
SEDACB	5
SEDAC	7
SED	9
SE	8



6.4. ábra

A Dijkstra-algoritmushoz hasonlóan a Bellman–Ford-algoritmus is működik irányítatlan gráfokra is, igaz, akkor több lehetséges viszonylatot kell megvizsgálni az egyes csúcsok esetében.

7. Hálótervezés

A tudományos kutatás és fejlesztés volumenének jelentős bővülése és bonyolultabbá válása, valamint a határidők rövidítésére és a fejlesztési költségek csökkentésére irányuló törekvés szükségessé tette olyan tervezési és irányítási módszerek kidolgozását, amelyek jobban igazodnak a tudományos kutatás és fejlesztés folyamatának sajátosságaihoz.

Olyan módszert kellett a tudománynak létrehoznia, amely alkalmas bonyolult rendszerek koordinálására, tervezésére, ellenőrzésére és irányítására. A kívánt módszereknek nemcsak a kutatás és a fejlesztés időszükségletének és költségfelhasználásának minél pontosabb felmérését, hanem a meghatározó tényezők optimalizálását is lehetővé kellett tennie a legkedvezőbb megoldás kiválasztása érdekében.[18]

Az 1950-es években az USA-ban olyan módszereket és rendszereket hoztak létre, amelyek úgynevezett hálódiagramok készítésén és felhasználásán alapulnak. Az első ilyen módszerek CPM[19] (*Critical Path Method,* kritikus út módszer) és PERT[20] (*Program Evaluation and Review Technique,* program-felülvizsgálati és -elemzési eljárás) elnevezéssel váltak ismertté. A CPM-módszert először építési munkák irányításánál, a PERT-módszert pedig hadászati célú berendezések kifejlesztésénél próbálták ki.

E módszerekhez számítógépek alkalmazása elvileg nem szükséges. A PERT tervezési rendszer kidolgozására, amelyet legelőször a Polaris atomtengeralattjáró-projektben használtak, kb. ötvenezer dollárt fordítottak. Ezen az összegen belül az algoritmusok és gépi feldolgozást célzó programok, valamint azok alkalmazása (gépi adatfeldolgozással együtt) kb. húszezer dollár, a módszer alkalmazásának ellenőrzése tizenötezer dollár, a módszer

---

[18] Abramov, Sz. A. – Marincsev, M. I. – Poljakov, P. D. (1966): *Hálódiagramos tervezési és irányítási módszerek.* Budapest, Közgazdasági és Jogi Könyvkiadó.

[19] Kelley, James E. Jr. – Walker, Morgan R. (1959): Critical-Path Planning and Scheduling. *1959 Proceedings of the Eastern Joint Computer Conference.* 160–173.

[20] Malcolm, D. G. – Roseboom, J. H. – Clark, C. E. – Fazar, W. (1959): Application of a Technique for Research and Development Program Evaluation. *Operations Research,* Vol. 7. No. 5. 646–669.

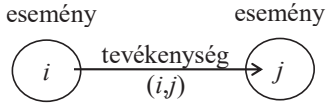
általánosítása tizenötezer dollár költséggel járt. Azonban már a fejlesztés új tervezési és irányítási módszerével szerzett első alkalmazási tapasztalatok is felülmúlták a várakozásokat. A szakemberek véleménye szerint csak a PERT bevezetésével sikerült a Polaris tengeralattjárók fejlesztési idejét 2-3 évvel csökkenteni. Pedig a fejlesztési terv tízezernél több eseményt tartalmazott, és hatezernél több cég vett részt a programban. A siker következtében a PERT-rendszert egyre szélesebb körben kezdték bevezetni.

A hálódiagramos módszereket először kétségtelenül hadicélok megoldására, a korszerű fegyverrendszerek létrehozására alkalmazták, az ott megoldandó műszaki problémák rendkívüli bonyolultsága miatt. Néhány év múlva azonban a hálódiagramos módszereket már polgári célokra is alkalmazni kezdték, például bonyolult épületek tervezésénél és építésénél, hírközlési vonalak építésénél, új gépek szerkesztésénél, új üzemek és számítóközpontok tervezésénél, iparvállalatok rekonstrukciójánál vagy új termékek bevezetésénél.

7.1. A hálódiagram szerkesztése

7.1.1. A háló elemei

A komplex termelési, kutatási, szervezési vagy akár beruházási feladat mindig több részfeladatból áll. Egy-egy ilyen részfeladatot a hálóban tevékenységnek nevezünk, amelyet egy kezdő és egy befejező pont határol. Ezek a pontok az események. A hálóban a tevékenységet nyíllal, a tevékenységet határoló eseményeket pedig körrel jelöljük (7.1. ábra).



7.1. ábra

Általában egy tevékenység befejező eseménye egyben a következő, kapcsolódó tevékenység induló eseménye is. Kivétel a háló első és utolsó tevékenysége, amelyeknél a megelőző, illetve következő tevékenység hiányzik.

A háló néha rendkívül sok tevékenységből áll, és ezért minden egyes tevékenységet valamilyen módon jelölni kell. A hálótervezésnél ezt az események

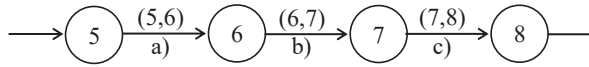
sorszámozásával oldjuk meg. A sorszámozásnál egy szabályt kell szigorúan betartani: minden tevékenységnél az (i,j) számozást úgy kell végrehajtani, hogy $i < j$, vagyis a tevékenységet jelző nyíl a kisebb sorszámú eseménytől a nagyobb sorszámú esemény felé mutasson. A háló kezdő eseményének általában a nulla sorszámot, míg a befejező eseménynek a háló legnagyobb sorszámát adjuk. Ha egy hálóban (n+1) esemény van, akkor a háló befejező pontjának az n sorszámot kell kapnia.

A 7.1. ábrában levő tevékenységet jelentő nyilat (a háló felépítése után) már nem a tevékenység rövid leírásával (például szerelés vagy szerkesztés stb.) nevezzük meg, hanem a tevékenység kezdő és befejező eseményének számával.

E szabályok alkalmazására nézzük a következő példát. Egy hálón belül az alábbi tevékenységek következnek sorrendben egymás után:

a) csövek helyszínre szállítása,
b) csővezetékek szerelése,
c) nyomáspróba.

Ezt a három tevékenységet a 7.2. ábrán mutatjuk be.



7.2. ábra

A nyíl alakjával kapcsolatosan meg kell jegyezni: nem szükséges, hogy mindig egyenes legyen. Összetettebb hálónál gyakran elkerülhetetlen, hogy a nyilak irányát megtörjük, néha éppen ez biztosítja az áttekinthetőséget.

7.1.2. A hálótervezés logikai szabályai

A háló szerkesztésének alapvető szabálya az, hogy a hálók a végrehajtásról egyértelmű tájékoztatást adjanak. A feladatok végrehajtásánál több lehetőséggel kell számolni:

- az egyes feladatok egymás után hajthatók végre,
- az egyes tevékenységek befejezése után egyszerre több tevékenység kezdhető meg, ezek párhuzamosan folyhatnak.

Ennek megfelelően a hálótervezésnél az összes részfeladatot, résztevékenységet úgy kell összeilleszteni, ahogyan azok valóban egymás után következnek, vagy esetleg egymástól függetlenül párhuzamosan folynak. A tevékenységeket valamilyen rendezőelv alapján kell szétválasztani, amely egyértelműen rávilágít az egyes tevékenységek kapcsolatára. Az alkalmazott rendezőelvek az alábbiak:

- technológiai rendezőelv,
- időbeli rendezőelv.

Logikai tervezés szempontjából a tevékenységek a következőképpen osztályozhatók:

- egymástól független tevékenységek,
- egymástól függő tevékenységek, amelyek lehetnek:
  - egyidejű tevékenységek,
  - megelőző tevékenységek,
  - követő tevékenységek.

7.1.3. A hálótervezés geometriai szabályai

A hálótervezés geometriai szabályai az alábbiak.

- A háló összefüggő, zárt rendszert alkot.
- A háló véges számú eseményből és tevékenységből áll, tehát a háló véges kiterjedésű.
- A háló irányított, időben lefolyó, mert a nyilak a korábbi időponttól a későbbi időpontig mutatnak.
- A hálónak hurokmentesnek kell lennie. Hurok olyankor képződik, ha a nyilak sorrendje önmagába visszatérő, zárt rendszert alkot. A hurokban a megelőző és követő tevékenységek nem különböztethetők meg.
- A folyamatokban a sorrendiséget mindig tisztázni kell, ezért a hálóban alternatív lehetőséget tervezni nem szabad. A hálótervezés egyik fő alkalmazási területe a műszaki fejlesztés, ahol olyan tevékenységek időbeni lefolyását is meg kell tervezni, amelyeknek műszaki megoldása még nem tisztázott. Hálótervezéssel határozható meg, hogy az adott problémát milyen időre kell vagy lehet megoldani. Az alternatív megoldások nem építhetők be a hálóba.
- A tevékenységeket ábrázoló nyilak metszhetik egymást.
- A háló alakja lényegtelen.

Ez utóbbival kapcsolatban meg kell említeni, hogy a háló előzetes elkészítése után szokásos úgynevezett topológiai rendezést végrehajtani, amelynek célja lehet:

- a háló alakjának szépítése,
- a tevékenységfelelős hozzárendelésének biztosítása,
- időarányos háló készítése.

7.1.4. A hálószerkesztés technikája

A hálók megszerkesztésénél célszerű bizonyos szerkesztési fogások alkalmazása.

Abban az esetben, ha egyes tevékenységek megkezdhetők anélkül, hogy az előző tevékenység befejeződött volna, az előző tevékenységet résztevékenységekre kell bontani.

A logikai tervezésnél be kell vezetnünk egy fogalmat, az úgynevezett látszattevékenység fogalmát. Tulajdonsága az, hogy nincs időszükséglete, a hálótervezésben szaggatott vonallal jelöljük. A látszattevékenység kizárólag logikai jellegű korlátozó tevékenység, ami arra vonatkozik, hogy az általa megadott logikai kapcsolat teljesítésétől teszi függővé egy további munka megkezdését. A látszattevékenység használatára általában a következő esetekben kerül sor:

- tevékenységek részfeladatokra bontásánál és átlapolásánál,
- két vagy több tevékenység ugyanazzal az eseménnyel kezdődik és fejeződik be,
- párhuzamosan folyó tevékenységek logikai kapcsolatának kifejezésénél,
- előfeltételek jelölésénél.

Nagy, komplikált hálóterveknél szokásos úgynevezett előzetes tevékenységet is tervezni. Ez felöleli mindazokat az adminisztratív teendőket, amelyeket a munka megkezdése előtt el kell végezni. Hullámvonallal szokásos jelölni. Az időtervezésnél ezt a tevékenységet nem veszik figyelembe.

A hálótervekben az egyes események jelentősége nem azonos. Az úgynevezett mérföldkövek módszere bizonyos eseményeknek kitüntetett fontosságot biztosít, ami abban nyilvánul meg, hogy a kitüntetett esemény bekövetkezéséig a munkák meghatározott szakasza lezárul, és más jellegű szakasz végrehajtása kezdődik meg. A módszernek mind az áttekintés

biztosítása, mind a később tárgyalásra kerülő határidőkérdés szempontjából van jelentősége.

Komplex terveknél célszerű részhálók kiemelése. Ekkor a fő koordinálást szolgáló komplex hálóban a részháló mint egyetlen tevékenység szerepel.

7.1.5. A hálószerkesztés folyamata

A háló megszerkesztésére több módszer alakult ki, amelyek az alábbiak:

- A háló szerkesztése kezdhető az első eseményből kiindulva.
- A háló szerkesztése kezdhető az utolsó eseményből kiindulva. Ez az úgynevezett retrográd módszer. Alapvető követelmény ugyanis a hálótervek készítésénél, hogy két eseményt csak egyetlen tevékenység köthet össze. A visszafelé történő szerkesztés célszerűsége abban mutatkozik meg, hogy általában könnyebb mindazokat a tevékenységeket felmérni, amelyek egy tevékenységet megelőznek, mint egy komplikált hálóban az eseményt követő összes következményt feltárni.
- A háló szerkesztése végezhető úgynevezett vegyes módszerrel. Ebben az esetben a komplex háló szerkesztése végezhető például az első eseményből kiindulva, míg a részlethálók szerkesztése az utolsó eseményből kiindulva történik.
- A háló szerkesztése kezdődhet valamely központi eseményből (például mérföldkőből) kiindulva, előre, illetve hátrafelé haladva.

7.1.6. A hálótervezés munkamenete

A hálótervezési munkát az alábbi lépésekben kell elvégezni:

- logikai tervezés,
- időtervezés,
- kapacitás tervezése,
- költségtervezés.

7.1.6.1. Logikai tervezés

A logikai tervezés az elvégzendő feladat áttekinthető, könnyen leellenőrizhető formában való ábrázolása. Olyan ábrázolási forma, amely az egész feladat lefolyásának, a munka menetének részletes leírását nyújtja, és egyben arra is alkalmas, hogy alapját képezze az időtervezésnek és a végrehajtás során a végrehajtás szervezésének és ellenőrzésének.

A logikai tervezés tehát a feladat elvégzéséhez szükséges komplex munkafolyamat diagramszerű ábrázolása, amely áttekintést nyújt az összetevők (részfolyamatok, tevékenységek) logikai, illetve technológiai összefüggéséről, kapcsolatáról és sorrendjéről. A tervezésnek ez a fázisa tehát a részmunkák, részfeladatok sorrendjére, egymásutániságára, a folyamatok technológiai és időbeni összefüggésének tisztázására hivatott.

A fentieknek megfelelően a tervezésnek ez a fázisa az alábbi lépésekben végzendő el, a logikai hálót az alábbiak szerint kell elkészíteni:

- A végrehajtandó feladat, a megvalósítandó cél meghatározása. A célnak különálló feladatnak kell lennie, amely pontosan körülhatárolható.
- A kitűzött cél érdekében elvégzendő összes teendő felmérése. Ebben a szakaszban végig kell gondolni a munkák megkezdésétől a befejezésig terjedő összes teendőt. Az elvégzendő feladatok teljes körű felmérése a hálótervezés egyik leglényegesebb feladata.

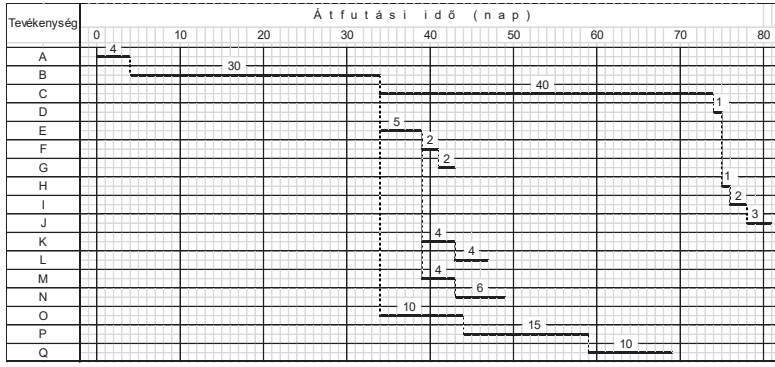
Ha bármely munkát kifelejtünk, veszélyeztetjük a hálótervezésre épülő időtervezés realitását, a végrehajtás szervezettségét, a nehézségek előzetes felismerését, és így nem határozhatók meg a szükséges intézkedések, de nem biztosítható az ellenőrzés sem. A munkák felmérésénél tisztázni kell a hálótervezésbe bevonandó területeket is:

- az elvégzendő feladatok, tevékenységek logikai (technológiai, időbeli) kapcsolatának meghatározása,
- a háló megszerkesztése, esetleg topológiai rendezése,
- az elkészített háló bírálata a hibák kijavítása érdekében.

A legjellemzőbb hibák a következők:

- egyes tevékenységek figyelmen kívül hagyása,
- egyes tevékenységek nem megfelelő kapcsolása.

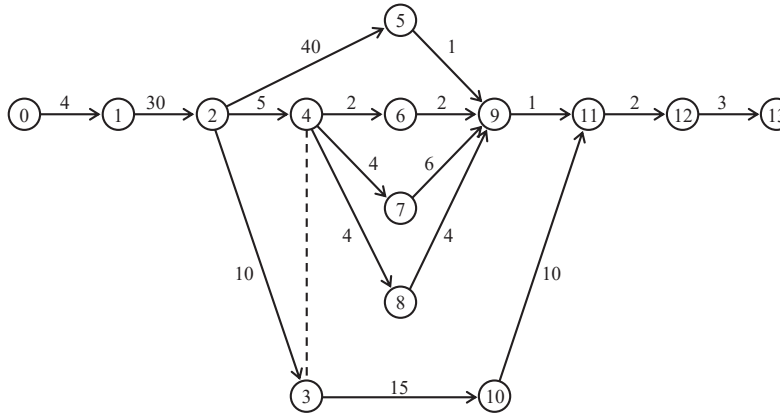
Az úgynevezett Gantt-diagramban ábrázolt helyzetkép alapján elkészíthető a logikai háló. A logikai háló elkészítéséhez első lépésként célszerű az elvégzendő feladatok logikai kapcsolatának meghatározását szolgáló táblázat elkészítése, amelyben fel kell tüntetni a tevékenységet közvetlenül megelőző és követő tevékenységeket. A gyakorlatban elterjedt az a módszer, hogy a közvetlenül megelőző és követő esemény számát adják meg. Ekkor nyilvánvalóan nem szükséges a tevékenységi jel alkalmazása, gyakran a tevékenység megnevezését is elhagyják, mert a kezdő és befejező esemény a tevékenységet egyértelműen megadja. Vegyük például, hogy egy laktanya felújítása 17 résztevékenységből álló folyamatra osztható. Rajzoljuk meg első lépésként a folyamat Gantt-diagramját (7.3. ábra).[21]



7.3. ábra

Jól láthatóak a diagramon, hogy az egyes tevékenységek milyen sorrendben követik egymást, illetve hogy mely tevékenységek befejezése szükséges más tevékenységek megkezdéséhez. Ezek alapján megrajzolhatjuk a tevékenységek logikai hálóját (7.4. ábra).

[21] Papp Ottó (1969): *A hálós programozási módszerek gyakorlati alkalmazása.* Budapest, Közgazdasági és Jogi Könyvkiadó.



7.4. ábra

Bonyolult, több tevékenységből álló folyamat szervezésénél a logikai háló elkészítése és az események beszámozása után készíthető el az úgynevezett tevékenységlista, amely a logikai háló felépítésére, a tevékenységekre, valamint ezek kapcsolatára egyértelmű felvilágosítást ad.

7.1.6.2. Időtervezés

Az előbbiekben a logikai háló felépítését tárgyaltuk anélkül, hogy az egyes tevékenységek időértékének meghatározásával foglalkoztunk volna. Megállapítottuk azt, hogy a logikai tervezés szolgál alapul az időtervezéshez.

A következőben az időtervezésre térünk át, annál is inkább, mivel a logikai háló nem alkalmas döntések kialakítására. Az időtervezés célja, hogy az egyes tevékenységek időtartamát felmérje, a tevékenységekhez időtartamot rendeljen, és meghatározza az egész feladat végrehajtásának átfutásiidő-igényét.

Mint az előzőekben már említettük, a látszattevékenységnek nincs időtartama, de az időtervezésben ennek ellenére rendkívül fontos szerepet tölt be. Az időtervezésre a gyakorlatban kétféle módszer terjedt el:

- határozott időtartamú tervezés (például CPM),
- határozatlan időtartamú tervezés (például PERT).

A kétféle tervezési módszer nemcsak az egyes tevékenységek időtartamának tervezésében, hanem a teljes folyamat átfutási idejének meghatározásában is különbözik. Ezért célszerű a két alapmódszert külön tárgyalni. A tervezés módszerétől függetlenül az időtervezés főbb lépései a következők:

- a hálóba felvett tevékenységek időtartamának meghatározása,
- a tevékenységsorok időtartamának, illetve a kritikus útnak a meghatározása,
- a tartalékidők számítása.

7.2. Határozott időtartamú tervezés (CPM-módszer)

A határozott időtartamú tervezés minden egyes tevékenységhez egyetlen meghatározott időtartamot rendel. Ebből következik célszerű alkalmazási területe: csak olyan helyen alkalmazható, ahol megfelelően kidolgozott műszaki normák állnak rendelkezésre. Ebben az esetben is felvetődik egy sor olyan probléma, amelynek megoldása alapját képezi a tervezés realitásának. Ezek a következők:

- a dolgozók munkateljesítményének figyelembevétele az egyes tevékenységek végrehajtásánál,
- nem munkaráfordítást, hanem az átfutási időt kell számítani, amely felveti az adott tevékenységen egyidejűleg dolgozó létszám megállapításának problémáját, különösen abban az esetben, ha párhuzamosan folyó tevékenységek azonos szakképzettségű dolgozókat igényelnek;
- az előkészítő tevékenységek átfutási idejének figyelembevétele,
- az anyagmozgatás, az előkészítés, segéd- és mellékfolyamatok időigényének, illetve átfutási idejének figyelembevétele.

Példánkban az egyes tevékenységek átfutási idejét megadtuk, a továbbiakban csak a teljes feladat időigényének, illetve átfutási idejének meghatározásával foglalkozunk.

Mint már említettük, a hálóban csomópont képződik, ha két vagy több tevékenység kezdő, illetve befejező eseménye közös. Ebből egyértelműen az is megállapítható, hogy adott esemény több úton érhető el a hálóban. Példánkban a lehetséges utak száma öt:

1. út: (0,1) (1,2) (2,5) (5,9) (9,11) (11,12) (12,13)
2. út: (0,1) (1,2) (2,4) (4,6) (6,9) (9,11) (11,12) (12,13)
3. út: (0,1) (1,2) (2,4) (4,7) (7,9) (9,11) (11,12) (12,13)

4. út: (0,1) (1,2) (2,4) (4,8) (8,9) (9,11) (11,12) (12,13)
5. út: (0,1) (1,2) (2,3) (3,10) (10,11) (11,12) (12,13)

Az összes út közül egyetlen érdekel bennünket, és ez a legnagyobb időszükséglettel rendelkező út. A feladat kivitelezésének időtartamát a sok út közül ugyanis az az út adja meg, amelyik a legnagyobb átfutási időt igényli, ez az úgynevezett kritikus út. A kritikus úton levő tevékenységeknél a legkisebb késés is az egész határidő eltolódását okozza.

Nagy, komplex háló esetében azonban az összes lehetséges útvonal áttekintése rendkívül bonyolult vagy gyakorlatilag lehetetlen lenne. Szükséges tehát olyan módszer, amelynek segítségével a kritikus út gyorsan és egyértelműen meghatározható. Mielőtt azonban a módszerek tárgyalására rátérnénk, néhány fogalommal és jelöléssel szükséges megismerkedni.

Időtartam. Jelölése: $y(i,j)$, az (i,j) tevékenység végrehajtásának időtartamát jelöli. $y(i,j) = y(j,i)$

Legkorábbi befejezési időpont. Jelölése: $t_f(i)$, amikorra az összes, i időpontban végződő tevékenységet be lehet fejezni.

Legkésőbbi kezdési időpont. Jelölése: $t_s(i)$, amikor egy, az i időpontban kezdődő tevékenységnek meg kell kezdődnie ahhoz, hogy a végső határidő tartható legyen. Két út különbsége adja, tehát $t_s(i)$ = (kritikus út) – (i-től a végpontig terjedő leghosszabb út). Természetesen a számításoknál ez a bonyolult definíció így nem alkalmazható, ezért mindig a követő esemény legkésőbbi kezdési időpontjából visszafelé történik a számítás.

A következőkben a kritikus út meghatározását az úgynevezett nyomonkövetési módszerrel fogjuk elvégezni. A nyomonkövetési módszer alkalmazásánál mind a $t_f(i)$, mind pedig a $t_s(i)$ értékeket meg kell határozni.

7.2.1. A legkorábbi befejezési időpont számítása

A $t_f(i)$ érték számításának meneténél az előbbi definícióra támaszkodunk, amely szerint a $t_f(i)$ a legkorábbi befejezési időpont, amikor az i időpontban végződő tevékenységeket be lehet fejezni, tehát az i eseményig tartó leghosszabb út.

A számítás menete:

- a kezdő eseménynél $t_f(0) = 0$
- egyszerű eseménynél a $t_f(i)$ értéket úgy számítjuk, hogy az i eseményt közvetlenül megelőző eseményhez tartozó $t_f(i–1)$ értékhez

hozzáadjuk az i eseményhez vezető $(i–1,i)$ tevékenység időértékét, $y(i–1,i)$-t. A példánkban így számíthatók az i = 1, 2, 3, 5, 6, 7, 8, 10, 12, 13 eseményekhez tartozó $t_f(i)$ értékek, ha sorban haladunk, és a megelőző eseménypontok értékeit már számítottuk. A számítási összefüggés tehát: $t_f(i) = t_f(i–1) + y(i–1,i)$.

- csomópont esetén a $t_f(i)$ értéket úgy számítjuk, hogy az i eseményt közvetlenül megelőző eseményekhez tartozó $t_f(a)$ értékekhez hozzáadjuk az i eseményekhez vezető (a,i) tevékenységek időértékét, és az összes lehetőséget figyelembe véve a számított legnagyobb érték lesz a $t_f(i)$ érték. A számítási összefüggés tehát: $t_f(i) = \max\{t_f(a) + y(a,i)\}$, minden elemi (a,i) tevékenységre.

A példánkban így számítható az i = 4, 9, 11 eseményekhez tartozó $t_f(i)$ érték, ha sorban haladunk, és az

i = 4 esemény esetén a $t_f(2)$ és a $t_f(3)$,
i = 9 esemény esetén a $t_f(5)$, a $t_f(6)$, a $t_f(7)$ és a $t_f(8)$,
i = 11 esemény esetén pedig a $t_f(9)$ és a $t_f(10)$ értéket már kiszámítottuk.

Nézzük meg, hogyan kell a számításokat a bemutatott példánkban konkrétan végrehajtani:

$t_f(1) = 4$
$t_f(2) = 4 + 30 = 34$
$t_f(3) = 34 + 10 = 44$

A $t_f(4)$ esemény legkorábbi befejezési időpontja a $t_f(2)$ és a $t_f(3)$ események figyelembevételével számítható. A $t_f(3)$ és $t_f(4)$ eseményeket azonban látszattevékenység köti össze, ami azt jelenti, hogy a harmadik és negyedik eseményt egy időpontban kell kezdeni. A 4. esemény legkorábbi befejezési időpontja tehát

a $t_f(2)$ eseményből $t_f(4) = 34 + 5 = 39$,
a $t_f(3)$ eseményből $t_f(4) = 44 + 0 = 44$,
azaz $t_f(4) = \max\{t_f(2) + y(2,4); t_f(3) + y(3,4)\} = \max\{34 + 5; 44 + 0\} = \max\{39; 44\} = 44$.

$t_f(5) = 34 + 40 = 74$
$t_f(6) = 44 + 2 = 46$
$t_f(7) = 44 + 4 = 48$
$t_f(8) = 44 + 4 = 48$
$t_f(10) = 44 + 15 = 59$

A $t_f(9)$ eseménynél csomóponthoz érkeztünk, tehát a csomópontba kapcsolódó összes esemény figyelembevételével kell a legkorábbi befejezés időpontját számítani:

a $t_f(5)$ eseményből $t_f(9) = 74 + 1 = 75$,
a $t_f(6)$ eseményből $t_f(9) = 46 + 2 = 48$,
a $t_f(7)$ eseményből $t_f(9) = 48 + 6 = 54$,
a $t_f(8)$ eseményből $t_f(9) = 48 + 4 = 52$,
azaz $t_f(9) = \max\{t_f(5) + y(5,9); t_f(6) + y(6,9); t_f(7) + y(7,9); t_f(8) + y(8,9)\} = \max\{74 + 1; 46 + 2; 48 + 6; 48 + 4\} = \max\{75; 48; 54; 52\} = 75$.

A 11. esemény szintén csomópont, így a már megismert módon végezzük el a számításokat:

a $t_f(9)$ eseményből $t_f(11) = 75 + 1 = 76$,
a $t_f(10)$ eseményből $t_f(11) = 59 + 10 = 69$,
azaz $t_f(11) = \max\{t_f(9) + y(9,11); t_f(10) + y(10,11)\} = \max\{75 + 1; 59 + 10\} = \max\{76; 69\} = 76$.

Már csak két esemény van hátra, de ezek egyszerű események, így a számításuk nem bonyolult:

$t_f(12) = 76 + 2 = 78$
$t_f(13) = 78 + 3 = 81$

Tehát az egész folyamat legkorábbi befejezésének ideje 81 nap.

7.2.2. A legkésőbbi kezdési időpont számítása

A legkésőbbi kezdési időpont a definíció szerint azt az időpontot jelenti, amikor legalább egy, az i időpontban kezdődő tevékenységnek meg kell kezdődnie ahhoz, hogy a határidő ne tolódjék el.

Számítása a befejező esemény $t_f(i)$ értékéből visszafelé történik úgy, hogy ebből az értékből levonjuk az i eseménytől a háló befejező eseményéig terjedő utak közül a leghosszabbat.

A számítás menete:

- A háló végpontjából indulunk ki. A befejező eseményből a $t_s(n) = t_f(n)$.
- Egyszerű eseménynél a $t_s(i)$ értéket úgy számítjuk, hogy az i eseményt közvetlenül követő eseményhez tartozó $t_s(i+1)$ értékből levonjuk az i eseményhez vezető $(i,i+1)$ tevékenység időértékét. A példánkban így

számíthatók az $i = 12, 11, 10, 9, 8, 7, 6, 5, 1, 0$ eseményekhez tartozó $t_s(i)$ értékek, ha hátulról előre haladunk, és a követő eseménypontok $t_s(i)$ értékeit már számítottuk. A számítási összefüggés tehát: $t_s(i) = t_s(i+1) - y(i,i+1)$.

- Csomópont esetén a $t_s(i)$ értéket úgy számítjuk, hogy az i eseményt közvetlenül követő eseményekhez tartozó $t_s(a)$ értékekből levonjuk az i eseményhez vezető (i,a) tevékenységek időegységét, és az összes lehetőséget figyelembe véve a számított legkisebb érték lesz a $t_s(i)$ érték. A számítási összefüggés tehát: $t_s(i) = \min\{t_s(a) + y(i,a)\}$, minden elemi (i,a) tevékenységre.

A példánkban így számítható az $i = 4, 3, 2$ eseményekhez tartozó $t_s(i)$ érték, ha hátulról előre haladunk, és

az $i = 4$ esemény esetén a $t_s(6)$, a $t_s(7)$ és a $t_s(8)$,

az $i = 3$ esemény esetén a $t_s(4)$ és a $t_s(10)$,

az $i = 2$ esemény esetén pedig a $t_s(3)$, a $t_s(4)$ és a $t_s(5)$ értéket már kiszámítottuk.

Határozzuk meg példánkban a legkésőbbi kezdési időpontokat a folyamat végétől ($i = 13$) kiindulva:

$t_s(13) = 81$
$t_s(12) = 81 - 3 = 78$
$t_s(11) = 78 - 2 = 76$
$t_s(10) = 76 - 10 = 66$
$t_s(9) = 76 - 1 = 75$
$t_s(8) = 75 - 4 = 71$
$t_s(7) = 75 - 6 = 69$
$t_s(6) = 75 - 2 = 73$
$t_s(5) = 75 - 1 = 74$

A 4. esemény legkésőbbi kezdési időpontjának meghatározása szempontjából csomópontnak minősül, így minden kapcsolódó eseményre vonatkozóan meg kell határozni az értéket, és közülük a legkisebb lesz a $t_s(4)$ értéke.

A $t_s(5)$ eseményből $t_s(4) = 73 - 2 = 71$,

a $t_s(7)$ eseményből $t_s(4) = 69 - 4 = 65$,

a $t_s(8)$ eseményből $t_s(4) = 71 - 4 = 67$,

azaz $t_s(4) = \min\{t_s(6) - y(4,6); t_s(7) - y(4,7); t_s(8) - y(4,8)\} = \min\{73 - 2; 69 - 4; 71 - 4\} = \min\{71; 65; 67\} = 65$.

A harmadik esemény szintén csomópont, így meghatározása a bemutatott módon történik:

a $t_s(10)$ eseményből $t_s(3) = 66 - 15 = 51$,

a $t_s(4)$ eseményből $t_s(3) = 65 - 0 = 65$,

azaz $t_s(3) = \min\{t_s(10) - y(3,10); t_s(4) - y(3,4)\} = \min\{66 - 15; 65 - 0\} = \min\{51; 65\} = 51$.

A második esemény is csomópont, számítása a következő:

a $t_s(5)$ eseményből $t_s(2) = 74 - 40 = 34$,

a $t_s(4)$ eseményből $t_s(2) = 65 - 5 = 60$,

a $t_s(3)$ eseményből $t_s(2) = 51 - 10 = 41$,

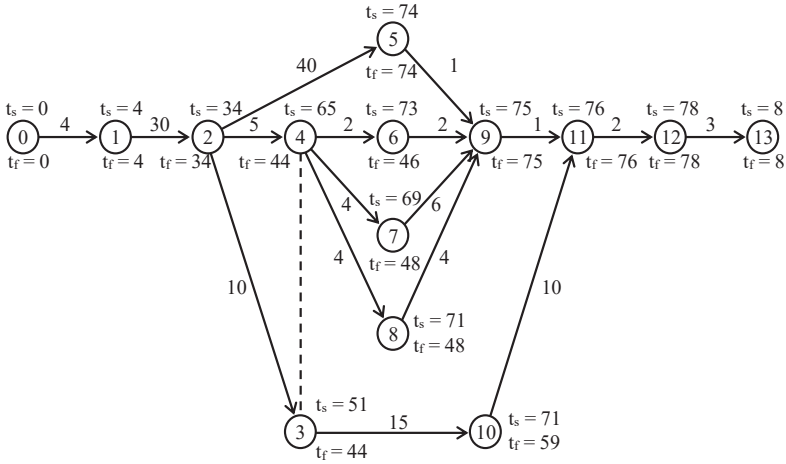
azaz $t_s(2) = \min\{t_s(5) - y(2,5); t_s(4) - y(2,4); t_s(3) - y(2,3)\} = \min\{74 - 40; 65 - 5; 51 - 10\} = \min\{34; 60; 41\} = 34$.

Még két eseményhez tartozó legkésőbbi időpontot kell meghatározni, de ezek egyszerű események, így a számítás nem okozhat problémát:

$t_s(1) = 34 - 30 = 4$

$t_s(0) = 4 - 4 = 0$

A háló egyes eseményeihez tartozó időértékeket a 7.5. ábra tartalmazza.



7.5. ábra

7.2.3. A kritikus út meghatározása

A nyomon követéssel számított értékek alapján, amelyeket a 7.5. ábra tartalmaz, érdekes összehasonlításokat tehetünk. A $t_f(i)$ és a $t_s(i)$ értékek viszonya kétféleképpen alakul:

$t_f(i) = t_s(i)$ az $i = 0, 1, 2, 5, 9, 11, 12, 13$ eseményeknél, és
$t_f(i) \neq t_s(i)$ az $i = 3, 4, 6, 7, 8, 10$ eseményeknél.

Ha az egyenlőségeket megvizsgáljuk, megállapíthatjuk, hogy azok az események, amelyeknél $t_f(i) = t_s(i)$, vagyis amelyeknél a legkorábbi befejezési időpontok és legkésőbbi kezdési időpontok megegyeznek, a kritikus úton fekszenek. Így tehát a kritikus út egyértelműen megállapítható. Megjegyezzük, hogy a kritikus utat az előzőekben úgy határoztuk meg, mint a hálórendszer kezdete és vége közötti leghosszabb utat. Ha tehát a hálórendszerben minden pontban vezető leghosszabb utat meghatároztuk, akkor egyúttal meghatároztuk a kritikus utat is.

7.3. Határozatlan időtartamú tervezés (PERT-módszer)

A határozatlan időtartamú tervezés abból indul ki, hogy a tevékenységek végrehajtásának ideje bizonyos határok között ingadozik, vagyis a véletlen befolyásolja. Ebből nyilvánvalóan következik, hogy a véletlen befolyással van a következő kapcsolódó tevékenységek legkorábbi kezdési időpontjára is.

Tulajdonképpen a folyamatnak ez az úgynevezett sztochasztikus jellege speciális, valószínűségszámítási alapokra épülő számítási módszer alkalmazását teszi szükségessé, ugyanakkor hangsúlyozzuk, hogy ugyanazokat a feladatokat kell megoldani, mint a határozott időtartamú tervezésnél. Például a PERT-módszernek az a lényege, hogy a tevékenységek időtartamának becslését úgy végzik el, hogy háromféle időből indulnak ki.

A legvalószínűbb idő (jele: *m*) az az időtartam, amely a becslést végző, a folyamatot szakmai szempontból kitűnően ismerő szakember szerint azonos körülmények között végezve a tevékenységét, a leggyakrabban fordul elő.

A legkedvezőbb idő (jele: *a*) az az idő, amelyet a becslő úgy választ, hogy a tevékenység ennél rövidebb idő alatt akkor sem hajtható végre, ha a körülmények különösen szerencsések (optimista becslés).

A legkedvezőtlenebb idő (jele: *b*) az adott tevékenység számára a legnagyobb időtartam (pesszimista becslés), amely alatt még különösen kedvezőtlen körülmények (például ismétlések, hibák) között is elvégezhető.

Természetesen a tervezés során háromféle idővel nem lehet számolni, ezért a három időből egyetlen időt határoznak meg, az úgynevezett várható időt (jele: t_e).

Az alkalmazott képlet:

$$t_e = \frac{a + 4m + b}{6}. \tag{7.1}$$

A t_e értéke tehát a becslésektől függ, de minél nagyobb a két szélsőséges idő (*a* és *b*) között az intervallum, annál nagyobb a becsült várható idő bizonytalansága. Ez utóbbiról úgy szerezhetünk információt, hogy meghatározzuk az idő várható értékéhez tartozó varianciákat (szórásnégyzeteket).

Béta-eloszlás esetén a szórásnégyzet:

$$\sigma_{t_e}^2 = \frac{(b-a)^2}{36}. \tag{7.2}$$

Ebből a szórás:

$$\sigma_{t_e} = \frac{b-a}{6}. \tag{7.3}$$

Amennyiben az egyes időtartamokat minden egyes tevékenységre különkülön megbecsültük az elmondottak szerint, meghatározható a kritikus út. A számítás a határozott időtartamú tervezésnél ismertetett módon történik, vagyis meghatározzuk a legkorábbi befejezési és a legkésőbbi kezdési időpontokat.

Az eltérés abban van, hogy az egyes tevékenységek időtartamai (amelyek a számítás alapját képezik), csak bizonyos valószínűséggel következnek be, és ennek megfelelően az egész háló, vagy annak egy részére vonatkozó átfutási idő is csak bizonyos valószínűséggel állapítható meg, amely természetesen függ az egyes tevékenységek időtartamának valószínűségétől.

Tehát gyakorlatilag azt kell megállapítani, hogy mekkora bizonytalanság mellett következik be a tevékenységsorozat utolsó tagjának befejezési időpontja, ha ismert a tevékenységsor minden egyes tagjának a szórása. A feladat az úgynevezett központi határeloszlás-tétel alapján oldható meg.

A tétel szerint minden olyan összeg, amely egymástól független, de ugyanazon eloszlásfüggvény törvényszerűségeit követő véletlen elemekből áll, jó közelítéssel normális, más néven Gauss-eloszlású.

A PERT-háló tevékenységelemeinek időtartamai összegzésével állapíthatjuk meg a hálóval jellemzett tevékenységsor határidőit. Ez az összeg kielégíti a központi határeloszlás-tétel feltételeit, így eloszlása normális. Ennek a normális eloszlásnak az átlagát (várható értékét) és szórását az egyes valószínűségi változók számtani átlagának, illetve szórásának összege adja. Tehát a PERT-háló esetén a teljes határidőre vonatkozó szórás nagyságát a kritikus úton fekvő tevékenységek szórásainak összegeként határozzák meg.

A tartalékidőket a PERT-módszer esetén is a megelőző tevékenység befejezésének és a követő tevékenység megkezdésének különbségeként számítják, de:

- csak a teljes tartalékidő számítható,
- a tartalékidőt eseményre orientáltan határozzák meg, vagyis azt vizsgálják, hogy a vizsgált eseménynek mekkora időintervallumon belül kell bekövetkeznie, a véghatáridő módosítása nélkül,
- meg kell határozni a tartalékidők eloszlását is.

8. Sorbanállás

A várakozó sorok a mindennapi élet közismert jelenségei. Várakozó sorok keletkeznek például pénztárak, határátkelőhelyek, rakodóhelyek előtt, benzinkutaknál, reptereken a csomagfeladásnál és a biztonsági ellenőrzésnél vagy városokban útkereszteződéseknél, különösen nagy csomópontoknál. A sorbanállás sokszor valóban csak nehezen küszöbölhető ki.

Olyan esetekben, amikor biztonságból vagy például gazdasági okokból a várakozást meghatározott érték alá akarjuk szorítani, törekedni kell olyan intézkedésekre, amelyek kidolgozásához a kiszolgálási elmélet nyújt segítséget.

8.1. A sorbanállási modellekről általában

A kiszolgálórendszer valamilyen szolgáltatást nyújtó kiszolgálóhely és az előtte kiszolgálásra várakozó fogyasztói egységek (felek, igények) sorának (például a rakodóhelyen rakodásra várakozó tehergépkocsik) elhelyezésére szolgáló várakozóhelyek együttese. A kiszolgálóhelyen egy vagy több kiszolgálócsatorna (például rakodógép) áll rendelkezésre. A várakozóhelyek száma (a várakozó sor hossza) korlátozott vagy korlátlan lehet.

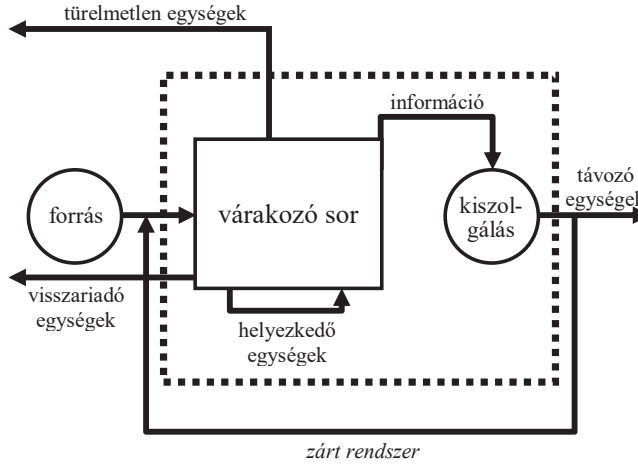
Strukturális szempontból vizsgálva zárt és nyílt kiszolgálórendszerek különböztethetők meg. Zárt kiszolgálórendszerekben nem változik a fogyasztói egységek száma, azaz a kielégítendő igények száma korlátozott. Nyílt kiszolgálórendszerekből a kielégített fogyasztói egységek távoznak, illetve oda állandóan új fogyasztói egységek érkeznek. A kielégítendő igények száma tehát ebben az esetben nagyon nagy, esetleg végtelen is lehet.

A kiszolgálási igények intenzitására mindkét esetben a fogyasztói egységek érkezései közötti időközök jellemzők. Zárt kiszolgálórendszerek esetében még az ugyanazon fogyasztói egység két érkezése közötti időköznek is jelentősége van. Az érkezési időközök lehetnek egyenlők; egyenlőtlenek, de meghatározottak; valamint egyenlőtlenek és nem meghatározottak (az időköz valószínűségi változó). Az érkezési folyamatot a fogyasztói egységek

érkezései közötti időközök eloszlásfüggvénye (például exponenciális, Erlang, Poisson stb.) szerint lehet csoportokba sorolni.

A fogyasztói egységek által igényelt kiszolgálási idő különbözősége, illetve azonossága miatt megkülönböztethető az egyforma (homogén) és a nem egyforma (inhomogén) fogyasztói egységek esete.

A várakozó sor azokból a fogyasztói egységekből tevődik össze, amelyek kiszolgálása nem kezdhető meg azonnal a rendszerben, hossza nulla és végtelen között változhat. A megengedett hosszúságú várakozó sor kialakulása után érkező igényeket a rendszer nem fogadja, elutasítja (8.1. ábra).



8.1. ábra

A várakozó sorra a fogyasztói egységek viselkedése is jellemző. Az úgynevezett türelmes egységek kivárják, amíg egy kiszolgálócsatorna szabaddá válik, az úgynevezett türelmetlen egységek viszont csak egy meghatározott ideig várakoznak, vagy pedig már a kiszolgálórendszerbe érkezésükkor döntenek (a már kialakult várakozó sor hosszától függően), hogy várakoznak, vagy kiszolgálás nélkül elhagyják a rendszert.

A kiszolgálócsatornák összességéből álló kiszolgálóhely strukturális szempontból soros, párhuzamos vagy vegyes elrendezésű, a kiszolgálócsatornák száma szempontjából pedig egy- vagy többcsatornás lehet.

A soros elrendezésű kiszolgálóhely kiszolgálócsatornáin a fogyasztói egységeknek az előírt sorrendben kell áthaladniuk, a kiszolgálásuk tehát

több szakaszból tevődik össze. Minden egyes csatorna előtt csak a megengedett hosszúságú várakozó sor alakulhat ki.

A párhuzamos elrendezésű kiszolgálóhely kiszolgálócsatornái az egyes fogyasztói egységeket teljes mértékben kiszolgálják. Általában közös várakozó sor alakul ki a kiszolgálóhely összes csatornája részére.

A kiszolgálás rendje a várakozó fogyasztói egységek soron következését szabja meg. Egy kiszolgálócsatorna felszabadulásakor ugyanis a várakozó sorban elhelyezkedő egységek közül különböző szempontok szerint lehet a következőt kiválasztani. Ebből a szempontból úgynevezett sordiszciplína szerinti és úgynevezett prioritásos diszciplína szerinti kiszolgálási rendet lehet megkülönböztetni, attól függően, hogy a fogyasztói egységeket a beérkezés sorrendjében szolgálják ki, vagy egyesek közülük előnyben részesülnek.

A prioritásos diszciplína szerinti kiszolgálás lehet abszolút és relatív prioritásos. Az abszolút prioritású rendszerben egy magasabb prioritású fogyasztói egység érkezésekor az éppen kiszolgált alacsonyabb prioritású egység kiszolgálását félbeszakítják, a magasabb prioritású egységet kezdik el kiszolgálni, és csak annak távozása után folytatják az alacsonyabb prioritású egység kiszolgálását. A relatív prioritású rendszerben a beérkező magasabb prioritású egységnek ki kell várnia az alacsonyabb prioritású egység már megkezdett kiszolgálásának befejezését, de annak végeztével azonnal a magasabb prioritású egység kiszolgálása kezdődik meg. Mindkét esetben addig nem folytatódik, illetve kezdődik el az alacsonyabb prioritású egység kiszolgálása, amíg van magasabb prioritású egység a rendszerben.

A kiszolgálási folyamatot is, az érkezési folyamathoz hasonlóan, a kiszolgálási idők eloszlásának törvényszerűsége szerint lehet csoportokba sorolni.

A tömegkiszolgálási elmélet alapja a sztochasztikus, véletlen jellegű folyamatok kialakulásának szabálytalansága, amely a fogyasztói egységek érkezésekor, várakozásukkor, valamint a kiszolgálásukkor jelentkezik, és amely a várakozó sor (például kielégítetlen anyagmozgatási igények) keletkezését megszabja.

A kiszolgálórendszerek különböznek egymástól a beérkezés és a kiszolgálás valószínűségi eloszlásának típusa, illetve egyenletessége, a kiszolgálóhely kiszolgálócsatornáinak száma és az egységek kiszolgálási rendje (sordiszciplína, illetve prioritásos diszciplína) szerint. Az egyes változatok matematikai vizsgálata is ennek megfelelően különféle módon mehet végbe. Általában azt feltételezik, hogy a kiszolgálandó egységek beérkezési időközei λ paraméterű exponenciális eloszlásúak (a beérkező igények Poisson-folyamat

szerint érkeznek), és a kiszolgálási időtartamok egymástól és az érkezési folyamattól független valószínűségi változók, azonos μ paraméterű exponenciális eloszlásúak vagy állandók.

Az érkezési és a kiszolgálási folyamatok jellegét a kiszolgálórendszer rövid, szimbolikus jellemzésével lehet kifejezni, amelyet David George Kendall (1918–2007) angol matematikus, statisztikus vezetett be, ezért utána Kendall-jelölésnek nevezünk. Az általános szimbolikus kifejezés: A/B/S, ahol A az érkezési folyamat, B a kiszolgálási folyamat típusát jelöli, S pedig a kiszolgálócsatornák számát mutatja.

A Kendall-jelölésben az A és a B helyén a következő, különböző folyamatokat jelentő betűket szokták alkalmazni:

M: Markov-folyamat, ami azt jelenti, hogy a rendszer „nem emlékezik", a jövőbeli állapota csak a jelenbeli állapotától függ, a múltbeli történésektől nem (még akkor sem, ha ténylegesen ismerjük azokat). Ezt Poisson-folyamattal szokás modellezni (minden Poisson-folyamat Markov-folyamat), amely a beérkezések egy olyan véletlen folyamata, amikor az egyes beérkezések közötti idők exponenciális eloszlást követnek.

G: általános *(general)* folyamat. Az egyes beérkezések közötti időközök μ várható értékkel és σ szórással jellemezhetők. Az egyes beérkezések függetlenek egymástól.

D: determinisztikus folyamat. A beérkezések közötti időtartamok állandóak.

A sorbanállási modellek alapfogalmainak könnyebb elsajátítása érdekében gondoljuk át a következő gyakorlati feladatot.

Adott egy cég, amelynek raktárából gépjárművek továbbítják az árut. A raktár rámpája mellett egyidejűleg N darab gépjármű fér el, azaz egy időben ennyi gépjármű megrakása történhet. A gépjárműveket, amikor érkeznek, illetve mielőtt a gyárat elhagyják, mérlegelik. Az egyszerűség érdekében feltételezhetjük, hogy külön mérleg áll rendelkezésre a beérkező és külön a távozó gépjárművek számára. A gépjárművek érkezése és megrakásuk ideje szabálytalan. Nyilvánvaló, hogy a leírt rendszer párhuzamosan kapcsolt többcsatornás sorbanállási rendszert alkot, ahol a kiszolgálás a rámpán történik, a kiszolgálócsatornák (vagy állomások) számát pedig a rámpa hossza határozza meg. Amennyiben a kiszolgálás csak a raktár ajtajánál lehetséges, a csatornák száma az ajtók számával egyezik meg.

A gépjárműveket érkezésük sorrendjében szolgálják ki, de más változat is elképzelhető. Ilyenek lehetnek: a gépjárműveket valamilyen szempont

(teherbírás, szállítandó áru, tulajdonjog stb.) szerint csoportosítva, csak meghatározott csatornán szolgálják ki, egyes gépjárműveket (például a sürgős szállítást lebonyolítókat) elsőbbségben részesítik (prioritás), de lehet a kiszolgálás tetszőleges, a beérkezési sorrendtől teljesen független is.

A kiszolgáló központba érkező egységek kívülről, úgynevezett forrásból jönnek. Az érkezők számától függően megkülönböztetünk korlátos és korlátlan forrású rendszereket. Például ha az árut bárki saját gépjárművével elszállíthatja, az érkező tehergépjárművek forrása korlátlan, abban az esetben viszont, ha azt kizárólag valamely fuvarozó vállalat, meghatározott számú gépjárműparkjával továbbíthatja, a gépjárművek forrása korlátos.

Az érkező gépjárművek először a gyár kapujánál várakoznak mérlegelésre. Itt a kiszolgálócsatorna a mérleg, a kiszolgálási idő a mérés ideje, beleértve a mérlegre hajtás és a távozás időszükségletét is. Ez egy egycsatornás rendszer. Ugyanez ismétlődik a gyár elhagyásakor is. Ha a gépjárművek gyárban töltött idejét vizsgáljuk, sorba kapcsolt csatornákat tartalmazó modellt kell felépítenünk, mégpedig olyat, melyben két egycsatornás rendszer fog közre egy többcsatornás párhuzamos sorbanállási rendszert.

Előfordulhat, hogy egyes gépjárművek nem tudnak hosszú ideig várakozni (más fontosabb szállítanivalójuk van, vagy ugyanezt a terméket máshol gyorsabban megkaphatják stb.), és kiválnak a sorból, amelyhez előzőleg csatlakoztak. Ilyenkor elhagyásos rendszerről van szó. Ha azonban az érkező gépjármű a hosszú várakozó sor láttán visszariad, és úgy dönt, hogy nem lép be a várakozók közé (esetleg már nincs is hely), akkor visszatorpanásos modellt kell alkalmaznunk. Lehet, hogy szemfüles gépjárművezetők kiugranak a sorból, és (feltéve, ha több várakozó sor van) egy másik sorhoz csatlakoznak. Az ilyen rendszert helyezkedésesnek nevezik.

Elképzelhető, hogy az érkező és távozó gépjárművek mérlegeléséhez nem áll külön-külön mérleg rendelkezésre. Ilyenkor a várakozó sorok ütköznek, interferálnak, ami tovább komplikálhatja a várakozó sor modelljét.

A sorbanállási modellekre vonatkozó általános ismeretek után ismerkedjünk meg először az egycsatornás, majd a többcsatornás tömegkiszolgálási rendszerekkel. Célunk most is az, hogy a módszerek gyakorlati alkalmazását mutassuk be, ezért az elméleti alapokkal részletesen nem foglalkozunk. Így csak az egyes rendszerekben meghatározandó paramétereket, illetve azok konkrét kiszámítási módját kívánjuk részletezni.

8.2. Egycsatornás tömegkiszolgálási rendszerek

Először is ismerkedjünk meg néhány olyan paraméterrel, amelyek mindegyik egycsatornás rendszerben megtalálhatók:

λ: érkezési ráta. Az időegység alatt érkező egységek számát jelenti. Dimenziója egység/idő.

μ: kiszolgálási ráta. Az időegység alatt kiszolgált egységek száma. Mértékegysége szintén egység/idő.

ϱ: forgalmi intenzitás. Ha $\lambda > \mu$, vagyis az időegység alatt érkező egységek száma meghaladja az időegység alatt kiszolgált egységek számát, korlátlan forrású érkezés esetén végtelen nagy és állandóan növekvő várakozó sor keletkezik. Ezért megköveteljük a $\varrho < 1$ feltétel teljesülését.

$P_0 = 1–\varrho$: annak a valószínűsége, hogy nem tartózkodik egység a rendszerben.

$P_n = \varrho^n(1–\varrho)$: annak a valószínűsége, hogy n darab egység tartózkodik a rendszerben.

8.2.1. Poisson-érkezésű, exponenciális kiszolgálási rendszer (M/M/1)

Az M/M/1 típusú kiszolgálási rendszer esetében egy kiszolgálócsatorna van, a beérkezések Poisson-folyamat szerintiek, és a kiszolgálási idők exponenciális eloszlásúak. A sorbanállási probléma megoldása során annak egyes tulajdonságai a következő módon számíthatóak.

A rendszerben tartózkodó egységek átlagos száma:

$$\bar{n} = \sum_{n=0}^{\infty} nP_n = \sum_{n=0}^{\infty} n\rho^n(1-\rho) = \frac{\rho}{1-\rho}. \tag{8.1}$$

Ha a rendszerben n darab egység tartózkodik, egycsatornás kiszolgálórendszerről lévén szó, egy egység kivételével, amelyiket éppen kiszolgálják, az összes többi a sorban áll, azaz $v = n–1$. A sor várható hossza, azaz a sorban álló egységek átlagos száma:

$$\bar{v} = \bar{n} - 1 + P_0 = \frac{\rho}{1-\rho} - \rho = \frac{\rho^2}{1-\rho}. \tag{8.2}$$

A várható sorbanállási idő:

(8.3) $$\bar{t}_v = \frac{\bar{v}}{\lambda}.$$

Ha kihasználjuk, hogy egycsatornás kiszolgálórendszerről beszélünk, a számláló helyére behelyettesíthetjük a (8.2) egyenletet:

(8.4) $$\bar{t}_v = \frac{\bar{v}}{\lambda} = \frac{1}{\mu}\frac{\rho}{1-\rho} = \frac{\bar{n}}{\mu}.$$

Az átlagos rendszerben töltött idő:

(8.5) $$\bar{t}_w = \frac{\bar{n}}{\lambda}.$$

Az átlagos kiszolgálási idő:

(8.6) $$\bar{t}_k = \bar{t}_w - \bar{t}_v = \frac{1}{\mu}.$$

A legfontosabb paraméterek meghatározását már ismerjük, nézzük most meg, hogyan kell ezt alkalmazni a gyakorlatban.

Tegyük fel, hogy egy kocsimosóhoz 8 gépkocsi érkezik óránként. A mosóban egy nyolcórás műszak alatt 80 gépkocsit tudnak lemosni. Mennyi időre van szüksége átlagosan egy gépkocsi lemosásához, mekkora a sor várható hossza és a rendszerben töltött teljes idő várható értéke, ha a sorbanállási modell M/M/1-es rendszerű?

Nézzük először meg, melyek azok a paraméterek, amiket ismerünk. Látható, hogy a feladatban adott az érkezési és a kiszolgálási ráta:

$\lambda = 8$ gk/h

$\mu = 80/8$ gk/h $= 10$ gk/h

E két adat ismeretében meg tudjuk határozni a forgalmi intenzitást is:

$\varrho = \lambda / \mu = 8/10 = 0{,}8 < 1$

Mivel a forgalmi intenzitás egynél kisebb, nem fog végtelen hosszú sor feltorlódni.

A rendszerben tartózkodó egységek átlagos számát a (8.1) képlet segítségével tudjuk megmondani:

$$\bar{n} = \frac{\rho}{1-\rho} = \frac{0{,}8}{1-0{,}8} = \frac{0{,}8}{0{,}2} = 4\,\text{gk}$$

Az átlagos kiszolgálási idő a (8.6) alapján az átlagos rendszerben töltött idő (8.5) és a várható sorbanállási idő (8.4) különbsége:

$$\bar{t}_w = \frac{\bar{n}}{\lambda} = \frac{4}{8} = 0{,}5\,\text{h} = 30\,\text{perc}$$

$$\bar{t}_v = \frac{\bar{n}}{\mu} = \frac{4}{10} = 0{,}4\,\text{h} = 24\,\text{perc}$$

Mindezek ismeretében már egyértelműen adódik az egy gépkocsira jutó átlagos kiszolgálási idő értéke:

$$\bar{t}_k = \bar{t}_w - \bar{t}_v = 30 - 24 = 6\,\text{perc} = \frac{1}{10}\,\text{h/gk} = \frac{1}{\mu},$$

vagyis átlagosan egy gépkocsit 6 perc alatt tudnak lemosni.

8.2.2. Poisson-érkezésű, tetszőleges eloszlású kiszolgálási rendszer (M/G/1)

Az M/G/1 típusú kiszolgálási rendszer esetében egy kiszolgálócsatorna van, a beérkezések Poisson-folyamat szerintiek, és a kiszolgálási idők adott μ várható értékű és σ szórású (ismeretlen eloszlású) valószínűségi változó szerintiek. Ilyen esetben a rendszerben található egységek számát az úgynevezett Kendall-képlettel határozzuk meg. Ez az eset a gyakorlatban többször előfordul.

A levezetés mellőzésével a rendszerben tartózkodó egységek száma a következők szerint írható fel:

$$\bar{n} = \rho + \frac{\rho^2 + \lambda^2\sigma^2}{2(1-\rho)}, \tag{8.7}$$

ahol σ a kiszolgálási idők szórását jelenti.

A várakozó sor átlagos hossza:

$$\bar{v} = \bar{n} - 1 + P_0. \tag{8.8}$$

Az $\bar{n}$ és a P_0 értékének behelyettesítésével már konkrétan meghatározhatjuk a várakozó sor átlagos hosszát:

$$\bar{v} = \rho + \frac{\rho^2 + \lambda^2\sigma^2}{2(1-\rho)} - 1 + (1-\rho) = \frac{\rho^2 + \lambda^2\sigma^2}{2(1-\rho)}. \tag{8.9}$$

Az átlagos sorbanállási idő:

$$\bar{t}_v = \frac{\bar{v}}{\lambda}. \tag{8.10}$$

A rendszerben tartózkodás átlagos ideje és az átlagos kiszolgálási idő meghatározása azonos az előző alpontban megismerttel, azaz a (8.5) és (8.6) kifejezések alapján számítható.

Oldjunk meg most egy feladatot a módszer gyakorlati alkalmazásának szemléltetésére!

Egy autópálya fizetőkapujához óránként átlagosan 25 gépjármű érkezik Poisson-folyamat szerint. Egy fizetés átlagosan 1,5 percig tart, de a kiszolgálás eloszlása ismeretlen, csak a szórását ismerjük: $\sigma = 0{,}08$ óra. Hány gépkocsi lesz átlagosan a fizetőkapunál, illetve mennyi ideig kell a gépkocsiknak átlagosan sorba állni?

Első lépésként most is azt vizsgáljuk, milyen adatokat ismerünk. Az érkezési ráta egyértelműen adott:

$\lambda = 25$ gk/h

A kiszolgálási rátát is meghatározhatjuk, hiszen tudjuk azt, hogy átlagosan 1,5 percig tart egy fizetés. Ezt figyelembe véve a kiszolgálási ráta értéke a következő:

$\mu = 1/1{,}5 = 2/3$ gk/perc

$\mu = 60/1{,}5 = 40$ gk/h

A forgalmi intenzitás a λ és a μ ismeretében már egyszerűen meghatározható:

$\varrho = \lambda / \mu = 25/40 = 0{,}625 < 1$

Ha megvizsgáljuk a rendszerben tartózkodó egységek meghatározásának (8.7) képletét, látható, hogy minden tényezője ismert már:

$$\bar{n} = \rho + \frac{\rho^2 + \lambda^2\sigma^2}{2(1-\rho)} = 0{,}625 + \frac{0{,}625^2 + 25^2 0{,}08^2}{2(1-0{,}625)} = 6{,}48.$$

Tehát átlagosan 6,48 gépkocsi tartózkodik a rendszerben. Meg kell még határoznunk az átlagos sorbanállási időt, de ennek számításához meg kell határozni a (8.9) kifejezés alapján a sorban várakozó gépkocsik átlagos számát is:

$$\bar{v} = \frac{\rho^2 + \lambda^2\sigma^2}{2(1-\rho)} = \frac{0{,}625^2 + 25^2 0{,}08^2}{2(1-0{,}625)} = 5{,}85.$$

Ezt felhasználva az átlagos sorbanállási idő a (8.10) alapján a következő lesz:

$$\bar{t}_v = \frac{\bar{v}}{\lambda} = \frac{5{,}85}{25} = 0{,}234.$$

Vagyis a gépkocsik átlagosan 0,234 órát, azaz 14,05 percet állnak sorba, amíg a fizetőkapuhoz beállhatnak.

8.2.3. Poisson érkezésű, determinisztikus kiszolgálási rendszer (M/D/1)

Az M/G/1-es rendszernél láttuk, hogy a rendszerben lévő egységek száma λ és μ adott értékei mellett a szórással növekszik. A rendszerben lévő egységek száma a minimumot akkor éri el, ha $\sigma = 0$, vagyis a kiszolgálási idő értéke állandó. Konstans kiszolgálási idő mellett a rendszerben tartózkodó egységek száma a következő:

(8.11)
$$\bar{n} = \rho + \frac{\rho^2}{2(1-\rho)}.$$

Az előző alfejezetben megismerthez képest a várakozó sor átlagos hosszának meghatározása is változik:

(8.12)
$$\bar{v} = \frac{\rho^2}{2(1-\rho)}.$$

Az átlagos sorbanállási idő, az átlagos rendszerben töltött idő és az átlagos kiszolgálási idő meghatározása azonos az M/G/1-es rendszernél bemutatottal, így ezt most nem részletezzük.

Nézzük meg, hogyan lehet ezt a módszert a gyakorlatban alkalmazni!

Egy rakodógép 10 perc alatt rak meg egy gépjárművet. A rakodóhelyre Poisson-eloszlás szerint 12 percenként érkeznek a gépjárművek. Mekkora

egy gépjármű átlagos várakozási vesztesége, ha a gépjárművek veszteglési költsége 20 €/óra/gépjármű?

A feladatban adott az érkezési és a kiszolgálási ráta, amelyekből a forgalmi intenzitás azonnal számítható:

$\lambda = 5$ gk/h

$\mu = 1$ gk / 10 perc = 6 gk / 60 perc = 6 gk/h

$\varrho = \lambda / \mu = 5/6 = 0{,}8\dot{3} < 1$

A forgalmi intenzitás ismeretében meghatározzuk a várakozó sor átlagos hosszát, mivel az átlagos várakozási idő kiszámításához erre az adatra is szükségünk lesz:

$$\bar{v} = \frac{\rho^2}{2(1-\rho)} = \frac{{}^{25}\!/_{36}}{2(1-{}^{5}\!/_{6})} = \frac{25}{12} = 2{,}08\dot{3}.$$

Az átlagos várakozási idő:

$$\bar{t}_v = \frac{\bar{v}}{\lambda} = \frac{2{,}08\dot{3}}{5} = 0{,}41\dot{6}\,\text{h}.$$

A várakozási veszteség: $0{,}41\dot{6}$ h·20 €/h = $8{,}\dot{3}$ €, vagyis egy gépjármű átlagos várakozási vesztesége $8{,}\dot{3}$ € lesz.

8.3. Többcsatornás tömegkiszolgálási rendszerek

Az egycsatornás tömegkiszolgálási rendszereknél láthattuk, hogy sztochasztikus érkezés és kiszolgálás esetén az esetek túlnyomó többségében várakozó sorral állunk szemben. A várakozás egyértelmű, hogy pluszkiadással jár. Az operációkutatás egyik lényeges feladata a kiszolgálóhelyek üzemeltetésének és a várható sorbanállás miatt jelentkező veszteségek egybevetése, illetve a kiszolgálócsatornák számának optimális meghatározása.

Egyértelmű, hogy több csatorna mellett gyorsabb a kiszolgálás is, de csekély számú érkezés esetén nagy a valószínűsége az esetleges kihasználatlanságnak is.

Tételezzük fel, hogy a rendszerbe Poisson-folyamat szerint érkeznek az egységek, a kiszolgálási idők pedig exponenciális eloszlásúak. Így a rendszer vizsgálatánál ugyanazok a tényezők szerepelnek, amelyeket az egycsatornás rendszereknél megismertünk. Az egyes tényezők jelentését azonban

a rendszernek megfelelően kell értelmezni. A beérkezési ráta (λ) és a kiszolgálási ráta (μ) dimenziója egység/idő. Hányadosuk most is a forgalmi intenzitást határozza meg, de ellentétben az egycsatornás rendszerrel, ahol a $\varrho < 1$ feltételt követeltük meg, most a $\varrho < S$ feltételnek kell teljesülnie, ahol S a kiszolgálócsatornák számát jelenti. Ha ez a feltétel nem teljesülne, a várakozó sor hossza minden határon túl növekedne.

A továbbiakban csak a korlátlan forrású, S csatornás sorbanállási rendszerrel (M/M/S) foglalkozunk.

A λ, μ és ϱ értékeinek meghatározása az egycsatornás rendszerekhez képest nem változott. Eltérés mutatkozik azonban a P_0 és P_n értékeinek meghatározásánál. Nézzük meg, hogy M/M/S rendszer esetében hogyan alakulnak ezen paraméterek értékei. A levezetést most is mellőzzük, csak a végeredményt közöljük.

(8.13)
$$P_0 = \left[\frac{\rho^S}{S!} \frac{1}{\left(1 - \frac{\rho}{S}\right)} + \sum_{k=0}^{S-1} \frac{\rho^k}{k!} \right]^{-1}$$

(8.14)
$$P_n = P_0 \frac{\rho^n}{n!}$$

A rendszerben várakozók átlagértékét a P_0 és P_n ismeretében már meg tudjuk határozni:

(8.15)
$$\bar{v} = \sum_{n=S+1}^{\infty} (n - S) P_n = \frac{\rho^{S+1}}{S\, S! \left(1 - \frac{\rho}{S}\right)^2} P_0$$

Az átlagos sorbanállási idő meghatározását is végre tudjuk hajtani a $\bar{v}$ ismeretében:

(8.16)
$$\bar{t}_v = \frac{\bar{v}}{\lambda} = \frac{\rho^S}{\mu\, S\, S! \left(1 - \frac{\rho}{S}\right)^2} P_0$$

Nézzük meg, milyen összefüggés áll fenn az $\bar{n}$ és $\bar{v}$ között:

(8.17) $$\bar{n} = \bar{v} + S - \Psi = \bar{v} + \Xi$$

ahol Ψ az el nem foglalt csatornák átlagos száma, Ξ az elfoglalt csatornák átlagos száma, $\bar{n}$ a rendszerben átlagosan tartózkodó egységek száma.

Oldjunk most meg M/M/S rendszerrel egy konkrét feladatot!

Egy rakodógéphez Poisson-eloszlás szerint 2 perc alatt átlagosan 8 konténer érkezik. A kiszolgálás exponenciális eloszlású, átlagos ideje 0,5 perc. Hány csatornát kell üzemeltetni, ha azt akarjuk, hogy a várakozási veszteség 10 €/konténernél kisebb legyen? Egy konténer egy perc várakozási ideje 5 € veszteséget jelent.

Először írjuk fel azokat a tényezőket, amelyek a feladatban egyértelműen meg vannak határozva:

$\lambda = 4$ konténer/perc

$\mu = 2$ konténer/perc

$\bar{t}_k = 0{,}5$ perc

A csatornák számát úgy kell meghatározni, hogy az átlagos várakozási idő kisebb legyen 2 percnél (ezt jelenti a legfeljebb 10 € várakozási veszteség):

$\bar{t}_v < 2$ perc

Az alapadatok ismeretében meg tudjuk határozni a forgalmi intenzitást:

$\varrho = \lambda / \mu = 4/2 = 2$

Már említettük, hogy többcsatornás rendszereknél a forgalmi intenzitás értéke mindig kisebb a szükséges csatornák számánál. Ebből adódik, hogy a számításokat 3 kiszolgáló csatornát feltételezve végezzük el. Először meghatározzuk P_0 értékét a (8.13) egyenlet alapján:

$$P_0 = \left[\frac{\rho^S}{S!} \frac{1}{\left(1 - \frac{\rho}{S}\right)} + \sum_{k=0}^{S-1} \frac{\rho^k}{k!} \right]^{-1} = \left[\frac{2^3}{3!} \frac{1}{\left(1 - \frac{2}{3}\right)} + \frac{2^0}{0!} + \frac{2^1}{1!} + \frac{2^2}{2!} \right]^{-1} =$$

$$= \left[\frac{8}{6} 3 + \frac{1}{1} + \frac{2}{1} + \frac{4}{2} \right]^{-1} = \frac{1}{4+1+2+2} = \frac{1}{9}$$

A P_0 ismeretében már számítható a (8.15) kifejezés segítségével a sorban várakozó egységek átlagos száma:

$$\bar{v} = \frac{\rho^{S+1}}{S\,S!\left(1-\frac{\rho}{S}\right)^2} P_0 = \frac{2^{3+1}}{3 \cdot 3!\left(1-\frac{2}{3}\right)^2} \frac{1}{9} = \frac{8}{9}$$

Az átlagos várakozási idő (8.16) segítségével való meghatározásához is ismerünk minden paramétert:

$$\bar{t}_v = \frac{\bar{v}}{\lambda} = \frac{\frac{8}{9}}{4} = \frac{\rho^S}{\mu\,S\,S!\left(1-\frac{\rho}{S}\right)^2} P_0 = \frac{2^3}{2 \cdot 3 \cdot 3!\left(1-\frac{2}{3}\right)^2} \frac{1}{9} = \frac{2}{9}$$

Várakozási veszteség: 5 €/perc·0,2 2̇ perc = 1,1 1̇ €, tehát a kitűzött célt három csatorna üzemeltetésével teljesíteni tudjuk.

Előfordulhatott volna az is, hogy 3 csatorna esetén a feltétel nem teljesül. Ilyen esetben újra el kell végezni a számításokat, de már 4 csatornát feltételezve.

Bibliográfia

Abramov, Sz. A. – Marincsev, M. I. – Poljakov, P. D. (1966): *Hálódiagramos tervezési és irányítási módszerek.* Budapest, Közgazdasági és Jogi Könyvkiadó.

Jándy Géza (1971): *Operációkutatás.* Budapest, Műszaki Könyvkiadó.

Kaufmann, Arnold (1964): *Az optimális programozás.* Budapest, Műszaki Könyvkiadó.

Kaufmann, Arnold (1968): *Az operációkutatás módszerei és modelljei.* Budapest, Műszaki Könyvkiadó.

Krekó Béla (1966): *Lineáris programozás.* Budapest, Közgazdasági és Jogi Könyvkiadó.

Krekó Béla (1972): *Optimumszámítás.* Budapest, Közgazdasági és Jogi Könyvkiadó.

Papp Ottó (1969): *A hálós programozási módszerek gyakorlati alkalmazása.* Budapest, Közgazdasági és Jogi Könyvkiadó.

Ludovika Egyetemi Kiadó Nonprofit Kft.
Székhely: 1089 Budapest, Orczy út 1.
Kapcsolat: info@ludovika.hu

A kiadásért felel: Koltányi Gergely ügyvezető
Felelős szerkesztő: Karácsony Fanni
Olvasószerkesztő: Szabó Ilse
Tördelőszerkesztő: Fehér Angéla

ISBN 978-963-531-016-6 (nyomtatott)
ISBN 978-963-531-017-3 (pdf)

Több mint tíz év telt el azóta, hogy a Nemzeti Közszolgálati Egyetemen, pontosabban egyik elődintézményében, a Zrínyi Miklós Nemzetvédelmi Egyetemen egyetemi jegyzet készült az operációkutatás témakörében. De nemcsak a képzési struktúra ez idő alatt bekövetkezett változásai, hanem a mindennapi használat részévé vált újabb módszerek és alkalmazási területek bemutatásának szükségessége is indokolta egy korszerű jegyzet kiadását. Mivel új szakokon új tárgyak oktatása folyik, ezért a változó tantárgyi követelményekhez az oktatási anyagoknak is igazodniuk kell. Ennek a célnak kíván jelen jegyzet megfelelni, ugyanis célirányosan az NKE Hadtudományi és Honvédtisztképző Karon, a katonai logisztika BSc szakon oktatott Döntéselőkészítési módszerek kurzus tananyagának lefedésére készült.

A jegyzetben arra törekedtünk, hogy konkrét példákon mutassuk be az operációkutatási modellek katonai téren való alkalmazási lehetőségeit. Feltételeztük a mátrixalgebra, a valószínűségszámítás és a matematikai statisztika alapvető ismeretét. A leglényegesebb angol szakkifejezéseket az első szövegközi előfordulásukkor külön megjelenítjük.