Eötvös Loránd Tudományegyetem
Informatikai Kar

Új típusú pivot módszerek numerikus összehasonlítása a lineáris programozásban

Témavezető:
Dr. Illés Tibor
ELTE TTK
Operációkutatási tanszék

Készítette:
Nagy Adrienn
programtervező matematikus
hallgató

Budapest, 2007. június 1.

Köszönetnyilvánítás

Ezúton szeretném a köszönetemet kinyilvánítani dr. Illés Tibornak, a témavezetőmnek a szakdolgozat elkészítésében való segítségéért. Köszönettel tartozom még az összes tanáromnak, akik tanácsaikkal, javaslataikkal segítettek az egyetemi éveim alatt.

Kivonat

A dolgozat a lineáris programozás három legismertebb pivot alapú algoritmusát, a Dantzig-féle Szimplex algoritmust, a Monoton Build-Up Szimplex algoritmust, és a minimálindexes Criss-Cross algoritmust hasonlítja össze, összefoglalva az ezekhez tartozó elméleti hátteret, majd bemutatva az általunk egy saját fejlesztésű Matlab implementációval elért eredményeket.

Bár Dantzig 1947-ben mutatta be a Szimplex-algoritmusát, úgy tűnik a mai napig ez a leghatékonyabb pivot algoritmus a legtöbb nagy méretű probléma megoldására. A Criss-Cross algoritmus volt az első kombinatorikus pivot eljárás, amit Zionts [39] készített 1969-ben.

Az utóbbi években úgy tűnt, hogy a belső pontos eljárások megjelenése a pivot alapú eljárások hanyatlását okozza, de kiderült, hogy ez a két metódus tökéletesen kiegészíti egymást, mindkettőnek vannak előnyei és hátrányai a másikhoz képest.

Habár természetesen az ipari implementációk mint pl. a CPLEX sokkal hatékonyabban oldják meg a tesztfeladatokat, mint a saját implementációnk, mely ráadásul a feladatok standard alakját használja, így is a tesztfeladatok meglepően nagy részét képesek megoldani. Ezen belül is a Criss-Cross algoritmus adta meg a legkevesebb jó eredményt a feladatokra, míg a Szimplex és az MBU-Szimplex algoritmus nagyjából ugyanolyan számú feladatot volt képes megoldani.

Tartalomjegyzék

1. Bevezetés

Ez a dolgozat, a lineáris programozás három legismertebb pivot algoritmusát (Simplex, MBU-Szimplex, és Criss-Cross algoritmus) hasonlítja össze.

Az első fejezetben leírjuk a lineáris programozás rövid történetét, összefoglalva az algoritmusok kialakulását, fejlődését, majd röviden ismertetjük a szakdolgozatban használt jelöléseket.

A második fejezetben megadjuk azokat a fogalmakat, amelyeket a későbbiekben használni fogunk. Ismertetjük a lineáris egyenletrendszerekre vonatkozó definíciókat, alapvető tételeket, majd a mátrixokra vonatkozó jelöléseket. Végül pedig ismertetjük a Gauss-Jordan eliminációs algoritmust, amelyet majd használni fogunk a későbbi fejezetekben is.

A harmadik fejezet tartalmazza az algoritmusok konkrét, pszeudó kódos leírását, és az esetlegesen ehhez kapcsolódó fogalmakat, majd a fejezet végén egy rövidebb betekintést engedünk az MPS fájlformátumba is.

A negyedik fejezet tartalmazza az általunk készített implementációkban szereplő függvények leírását, hogy melyik eljárás melyik eljárást és függvényt hívja meg, használja a futása során, illetve szekvencia diagrammok segítségével bemutatjuk a konkrét futások egy részét.

Az ötödik fejezetben bemutatjuk azokat a tesztfeladatokat amelyeken az algoritmusokat teszteltük. Illetve ismertetjük az általunk elért eredményeket, táblázatos formában bemutatjuk, hogy mely algoritmus hány feladatot oldott meg, illetve melyek azok amelyekre nem futott.

Végül összefoglaljuk pár sorban az általunk elért eredményeket.

1.1. Történeti bevezetés

A lineáris egyenletrendszerek három változóval történő eliminálásnak megoldási módszere már a 9. fejezetben előfordul a "Kínai matematikusok matematikai műveszetében" i.e. 150 környékén. A Gauss-eliminációs eljárást, amelyik az egyik fő módszer a lineáris optimalizálásban Carl Freidrich Gauss (1777 - 1855) publikálta, és a később általánosított Gauss-Jordan eliminációs eljárást pedig a Földmérési Kézikönyvben (1873) W. Jordan (1842 - 1899) tette közzé. Az első alternatíva tételt azoknak a lineáris egyenletrendszereknek a karakterizációjával kapcsolatban, amelyeknek van megoldása A.

Capelli, Leopold Kronecker (1823 - 1891), és Eugene Rouché (1832 - 1910) fogalmazta meg az 1880-as években.

Az első jólismert modell, az irodalom szerint, - amely figyelembe vesz lineáris egyenlőtlenségeket is, mint korlátozó feltételeket, - a Fourier-féle mechanikai elv modellje. Ez a probléma képezte Farkas Gyula (1847 - 1930) számára a lineáris egyenlőtlenség rendszerek vizsgálatának kiinduló pontját. A lineáris egyenletrendszerek megoldhatóságát [10],[11] vizsgálta Farkas Gyula a 19. század vége felé. A lineáris egyenletrendszerekre vonatkozóan korai eredményeit foglalta össze német nyelű cikkében [12], és publikálta 1901-ben, amely azóta a leggyakrabban idézett cikk lett az optimalizálás történetében. Az ő híres alternatíva tétele, a Farkas lemma írja le a lineáris egyenlőtlenségrendszerek megoldhatóságát. A legismertebb számítási eljárást a különleges lineáris egyenlőtlenség rendszerek megoldására Jean Baptiste Joseph Fourier (1768 - 1830, ismert még ezenkívül a Fourier-sorokról is) írta le 1823-ban. Ezt az eljárást általánosította tetszőleges lineáris egyenlőtlenség rendszerekre Theodore Samuel Motzkin (1908 - 1970), és azóta ezt ismerik úgy, mint Fourier-Motzkin eliminációs eljárás [28].

Még jóval az igazán hatékony, a lineáris egyenlőtlenségrendszerek megoldására vonatkozó eljárások kifejlesztése előtt, Farkas híres alternatíva tétele megjelent magyarul is 1894-ben [10].

Vitathatatlanul a lineáris optimalizálás egyik atyja George Dantzig (1914 - 2005). Híres könyvében [8] összefoglal sok érdekes tényt, információt a Szimplex algoritmus és a lineáris programozás kialakulásáról.

A Szimplex algoritmust lineáris programozási problémák megoldására Dantzig fejlesztette ki 1947-ben.

Nem sokkal azután, hogy a Szimplex eljárás elérhetővé (ismertté) vált, elkezdtek az első alkalmazások kialakulni. A lineáris programozás első ipari alkalmazásai között volt az olajipar (beleértve az olaj előállítását, tisztítását, keverését, és szétosztását), az élelmiszeripar, hús csomagoló ágazat, a vas és acélipar, a fémfeldolgozás, a papírgyártás és a gazdasági irányítás.

Nem sokkal a Szimplex módszer felfedezése után, a kutatók elkezdtek alternatív pivot eljárásokat kidolgozni. Mégis, Dantzig Szimplex algoritmusa tűnik a leghatékonyabb pivot algoritmusnak a legtöbb nagy méretű probléma megoldására. Sokféle indexválasztási szabályt, és pivotálási eljárást alkalmaznak napjainkban. Ezeket T. Terlaky és S. Zhang [36] foglalta össze.

A Criss-Cross algoritmus [39], [36] volt az első tisztán kombinatorikus típusú pivot eljárás. Mivel Zionts képtelen volt az eljárásának a helyességét bebizonyítani, így ez az eredmény nem lett széles körben ismert. Egymástól függetlenül Chang [7], Terlaky [35], [34], [33], és Wang [38] is publikálta a Criss-Cross eljárást. A Criss-Cross algoritmus fejlődése megválaszolt egy sokáig nyitott kérdést, hogy egy lineáris programozási feladat megoldható-e egy fázisban avagy sem. Chang lineáris komplemetaritási feladatra, Terlaky lineáris megengedettségi-, lineáris programozási-, lineáris feltételes konvex kvadratikus-, lineáris komplementaritási- és irányított matroid feladatra, és Wang irányított matroid feladatra publikálta algoritmusát.

Bár a Szimplex algoritmus gyorsan jónak bizonyult a gyakorlatban, az elméleti bonyolultsága nyitott kérdés maradt 1970-ig, amikor Victor Kleene és George Minty készített egy példát, amelyben a minimálindexes Szimplex algoritmus exponenciális számú pivot műveletet hajt végre, egy alkalmasan választott csúcsból indulva, mielőtt megtalálja az optimális csúcsot. Ez a példa, a híres Klee-Minty kocka, egy némileg perturbált hiperkocka, amely nagyon hasznos elméleti eszköznek bizonyult azóta is. Bár különböző alternatív indexválasztási szabályt találtak ki, a legtöbb létrejöttének az oka egy exponenciális példa [4], [21], [24], [25], [27], [29] létezése vagy legalább feltételezett létezése.

Bár elméletileg nem polinomiális, a numerikus tapasztalatok szerint, a Szimplex algoritmus a gyakorlatban úgy működik, mint egy polinomiális algoritmus az esetek többségében, azonkívül az átlagos futási idő lineáris (vagy legfeljebb négyzetes). Ezt a gyakorlati tulajdonságot támasztja alá a cikkek széles skálája a várható pivot lépések számáról a Szimplex algoritmussal kapcsolatban [2], [3], [6], [37], bár ezek az eredmények szigorú valószínűségi kikötéseket feltételeznek a probléma input adataiban.

Az első polinomiális lineáris programozási algoritmus - 1978-ban került kifejlesztésre - a nemlineáris programozási technikák egyik speciális esete, amely megválaszolt egy régóta fennálló kérdést, miszerint a lineáris programozás a polinomiális lépésben megoldható, "könnyű problémák" osztályába tartozik-e avagy sem. Leonid G. Khachian (1953 - 2005) ellipszoid módszerének alapjait más szovjet matematikusok, Shor, Yudin és Nemrivskij fejlesztették ki. Annak ellenére, hogy az ellipszoid módszer elméletileg előnyös bonyolultságú, és kiváló a lineáris programozást használó elméletek számára, sajnos a használata nagyon eredménytelen a gyakorlatban.

A Narenda Karmarkar [20] által 1984-ben publikált algoritmus, az első polinomiális idejű algoritmus a lineáris programozásban, amely bebizonyította a hatékonyságát a gyakorlatban is. Hasonlóan az ellipszoid eljáráshoz, ennek az új, sokat ígérő kedvencnek a gyökerei is a nemlineáris programozáshoz kötődnek. Egy évvel a publikálás után, a Karmarkar eljárás ekvivalens formáját mutatták be klasszikus logaritmikus barrier eljárás néven, amit eredetileg a nemlineáris programozásban fedeztek fel [14]. Hamarosan megmutatták a kapcsolatot olyan klasszikus eljárásokkal, mint a Dikin féle Affin Skálázó Módszer [9], Fish (Nóbel díjas közgazdász) Logaritmikus Barrier Módszere [13], és Huard Középpont Módszere [16]. Ezek közül sok algoritmusról megmutatták, hogy lépésszáma a bemenő adatok egy polinomjával korlátozható [30].

Bár úgy tűnt, hogy a belső pontos eljárások megjelenése a pivot alapú eljárások hanyatlását okozza, de kiderült, hogy ez a két metódus tökéletesen kiegészíti egymást, mindkettőnek vannak előnyei és hátrányai a másikhoz képest. Az érvek és ellenérvek összefoglalását találjuk meg a [19]-ben. Ennek a két eljárástípusnak a versenye nagy fellendülést eredményezett mindkét algoritmusnál, az implementációk készítésében, és a teljesítmény növekedésben legalább két nagyságrenddel.

1.2. Jelmagyarázat

$\mathbf{a}, \mathbf{b}, ...$	a, b vektor
$A, M, ...$	A, M mátrix
$\{1, 2, ..., n\}$	1,2,...,n elemekből álló halmaz

2. Alapfogalmak

Először definiáljuk azokat az alapfogalmakat amelyek szükségesek lesznek majd az algoritmusok megértéséhez. Minden definíció után mutatunk egy példát is.

2.1. Definíció (Lineáris függetlenség). *Az* $\{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n\} \subset \mathbb{R}^m$ *vektorrendszert* lineárisan függetlennek *nevezzük, ha az* $\alpha_1\mathbf{a}_1 + \alpha_2\mathbf{a}_2 + ... + \alpha_n\mathbf{a}_n = 0$ *egyenlőségből következik, hogy* $\alpha_1 = \alpha_2 = ... = \alpha_n = 0$. *Nyilvánvaló, hogy az* $\mathbf{a} \neq \mathbf{0}$ *vektor önmagában lineárisan független rendszert alkot.*

Vagyis vektorok egy halmaza lineárisan független, ha nem léteznek olyan nem nulla együtthatók, amelyekkel, ha összeadjuk a vektorokat, nullvektort kapunk. Ez ekvivalens azzal, hogy a vektorrendszer egyik tagja sem fejezhető ki csupán a többi vektor segítségével.

2.2. Definíció (Generáló rendszer). *Az* $\{\mathbf{a}_i | i \in \mathcal{I}_G\}$ *(*$\mathcal{I}_G \subset \mathcal{I}$*) vektorhalmazt az* $\{\mathbf{a}_j | j \in \mathcal{I}\}$ generáló rendszer*ének nevezzük, ha bármely* $\mathbf{a}_j$ $j \in \mathcal{I}_N$ *(*$\mathcal{I}_N = \mathcal{I} \backslash \mathcal{I}_G$*) vektor előáll az* $\{\mathbf{a}_i | i \in \mathcal{I}_G\}$ *vektorok lineáris kombinációjaként.*

2.3. Definíció (Bázis). *Legyen adott az* $\{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n\} \subset \mathbb{R}^m$ *vektorok és egy* $\mathcal{I} = \{1, 2, ...n\}$ *indexhalmaz. Legyen adott továbbá egy olyan* $\mathcal{I}_B \subset \mathcal{I}$ *indexhalmaz, hogy az* $\{\mathbf{a}_i | i \in \mathcal{I}_B\}$ *vektorok lineárisan függetlenek és generáló rendszerét alkotják az* $\{\mathbf{a}_j | j \in \mathcal{I}\}$ *vektoroknak. Ekkor az* $\mathcal{I}_B$ *indexhalmazhoz tartozó vektorhalmazt* bázis*nak nevezzük.*

Vagyis a bázisban levő vektorokkal a többi vektor egyértelmű módon kifejezhető. Szokás a bázist úgy is definiálni, mint maximális elemszámú független részhalmazt.

2.4. Definíció (Rang). *Egy véges vektorrendszer* rang*ján tetszőleges bázisának elemszámát értjük.*

2.5. Definíció (Mátrix rang). *Egy* $M \in \mathbb{R}^{m \times n}$ mátrix rang*ján a lineárisan független vektorainak számát értjük,* $rang(M)$*-el jelöljük és* $rang(M) \leq min\{m, n\}$. *Ha* $rang(M) = min\{m, n\}$, *akkor a mátrixot teljesrangúnak szokás nevezni.*

Példa: Adott a következő 6 vektor, amelyek közül $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_4$ és $\mathbf{a}_6$ lineárisan függetlenek, és ezek a vektorrendszer egy bázisát alkotják.

1	2	2	0	7	0
1	3	0	0	5	0
0	4	3	1	-1	2
0	0	0	0	2	4
$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$

Ha a fenti vektorrendszert, mint mátrixot elnevezzük A-nak, akkor rang(A) = 4

2.6. Definíció (Bázistábla). *Legyen adott egy* $\{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n\} \subset \mathbb{R}^m$ *vektorhalmaz, és ennek egy* $\mathcal{I}_B$ *bázisa. Jelölje* $\mathcal{I}_N$ *a bázison kívüli elemek indexeit. Ekkor a* $T \in \mathbb{R}^{|\mathcal{I}_B| \times |\mathcal{I}_N|}$ *(rövid) bázistábla egy* t_{ij} *eleme, ahol* $i \in \mathcal{I}_B$ *és* $j \in \mathcal{I}_N$ *azt mondja meg, hogy az* a_j *nem bázisbeli elem bázisbeli előállításában mekkora az* a_i *együtthatója.*

Egy bázistáblát hosszúnak nevezünk, ha nemcsak az I_N-hez hanem a teljes I-hez tartozó oszlopok benne vannak. A bázistáblát szoktuk pivot táblának is nevezni.

Példa: A következő hosszú bázistáblában az $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_4, \mathbf{a}_6$ vektor van a bázisban, az $\mathbf{a}_3, \mathbf{a}_5$ pedig a bázison kívül.

	$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{e}_1$	$\mathbf{e}_2$	$\mathbf{e}_3$	$\mathbf{e}_4$
$\mathbf{a}_1$	1	0	6	0	11	0	3	-2	0	0
$\mathbf{a}_2$	0	1	-2	0	-2	0	-1	1	0	0
$\mathbf{a}_4$	0	0	11	1	6	0	4	-4	1	-1/2
$\mathbf{a}_6$	0	0	0	0	1/4	1	0	0	0	1/4

2.1. Tétel (Bázistranszformáció). *[15] Legyen* $\{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_n\}$ *a* V *vektortér bázisa,*

$$\mathbf{a} = \alpha_1 \mathbf{b}_1 + ... + \alpha_n \mathbf{b}_n. \tag{1}$$

Ha valamely $1 \leq i \leq n - re$ $\alpha_i \neq 0$*, akkor*

$$\{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_{i-1}, \mathbf{a}, \mathbf{b}_{i+1}, \mathbf{b}_n\} \tag{2}$$

ugyancsak bázisa a térnek.

Bizonyítás. Mivel $\alpha_i \neq 0$, ezért az $\mathbf{a} = \alpha_1 \mathbf{b}_1 + ... + \alpha_n \mathbf{b}_n$-ből $\mathbf{b}_i$ kifejezhető:

$$\mathbf{b}_i = -\sum_{j=1}^{i-1} \frac{\alpha_j}{\alpha_i} \mathbf{b}_j + \frac{1}{\alpha_i} \mathbf{a} - \sum_{j=i+1}^{n} \frac{\alpha_j}{\alpha_i} \mathbf{b}_j. \tag{3}$$

Legyen $\mathbf{x} = \sum_{j=1}^{n} \xi_j \mathbf{b}_j$ ahol ξ_j-k az előállításban szereplő súlyok. Ebbe a $\mathbf{b}_i$ vektort behelyettesítve kapjuk, hogy

$$\mathbf{x} = \sum_{j=1}^{i-1} \xi_j \mathbf{b}_j + \xi_i \left(-\sum_{j=1}^{i-1} \frac{\alpha_j}{\alpha_i} \mathbf{b}_j + \frac{1}{\alpha_i} \mathbf{a} - \sum_{j=i+1}^{n} \frac{\alpha_j}{\alpha_i} \mathbf{b}_j \right) + \sum_{j=i+1}^{n} \xi_j \mathbf{b}_j. \tag{4}$$

Rendezve (4)-t kapjuk, hogy

$$\mathbf{x} = \sum_{j=1}^{i-1} \left(\xi_j - \xi_i \frac{\alpha_j}{\alpha_i} \right) \mathbf{b}_j + \sum_{j=i+1}^{n} \left(\xi_j - \xi_i \frac{\alpha_j}{\alpha_i} \right) \mathbf{b}_j + \frac{\xi_i}{\alpha_i} \mathbf{a}. \tag{5}$$

Ezzel bebizonyítottuk, hogy (2) generátorrendszere a V vektortérnek. Bizonyítani kell még, hogy (2) lineárisan független vektorrendszer. Tegyük fel az ellenkezőjét. Legyenek $\alpha_1, \alpha_2, ..., \alpha_n$ nem mind nulla skalárok, melyekre

$$\gamma_1 \mathbf{b}_1 + ... + \gamma_{i-1} \mathbf{b}_{i-1} + \gamma_i \mathbf{a} + \gamma_{i+1} \mathbf{b}_{i+1} + ... + \gamma_n \mathbf{b}_n = 0. \tag{6}$$

Ha $\gamma_i = 0$, akkor az előző szerint $\{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_{i-1}, \mathbf{b}_{i+1}, \mathbf{b}_n\}$ lineárisan összefüggő vektorrendszer, ellentmondásban azzal, hogy $\{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_n\}$ lineárisan függetlenek. Ha $\gamma_i \neq 0$ akkor a fenti előállításból az $\mathbf{a}$ vektor kifejezhető:

$$\mathbf{a} = -\sum_{j \neq i} \frac{\gamma_j}{\gamma_i} \mathbf{b}_j. \tag{7}$$

Az (1) és (7) alapján

$$\left(\alpha_1 + \frac{\gamma_1}{\gamma_i} \mathbf{b}_1\right) + \left(\alpha_2 + \frac{\gamma_2}{\gamma_i} \mathbf{b}_2\right) + ... + \left(\alpha_{i-1} + \frac{\gamma_{i-1}}{\gamma_i} \mathbf{b}_{i-1}\right) + \tag{8}$$

$$\alpha_i \mathbf{b}_i + \left(\alpha_{i+1} + \frac{\gamma_{i+1}}{\gamma_i} \mathbf{b}_{i+1}\right) + ... + \left(\alpha_n + \frac{\gamma_n}{\gamma_i} \mathbf{b}_n\right) = 0$$

Ez pedig azt jelenti, hogy a $\{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_n\}$ vektorok előállítják a nulla vektort nem triviális módon, de ez azt jelentené, hogy a $\{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_n\}$ vektorok nem lineárisan függetlenek, ami ellentmond a bázis tulajdonságnak [15]. ■

Tehát a fenti, bázis transzformációra vonatkozó tételt úgy is megfogalmazhatjuk, hogy ha az $\mathbf{a}$ vektornak a $\{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_n\}$ bázisra vonatkozó i-edik koordinátája nem nulla, akkor a $\mathbf{b}_i$ bázisvektort $\mathbf{a}$-val kicserélve ismét bázist kapunk. Azt is mondhatjuk, hogy az $\mathbf{a} \neq \mathbf{0}$ vektor beépíthető a bázisba.

A fenti tétel bizonyítása alapján pivotáláskor a koordinátákat a következő módon lehet kiszámolni a fent definiált bázistáblában.

2.2. Tétel (Pivotálás). *Legyen adott az* $\{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n\} \subset V$ *vektorrendszer, ahol* V *egy vektortér. Ha* $t_{rs} \neq 0$ *akkor az* $\mathcal{I}_B$ *bázisban lévő* $\mathbf{a}_r$ *vektort kicserélhetjük a bázisban nem szereplő* $\mathbf{a}_s (s \in \mathcal{I}_N)$ *vektorral az alábbi módon:*

$$t'_{ij} = t_{ij} - \frac{t_{rj} t_{is}}{t_{rs}} \; i \in \mathcal{I}'_B, i \neq s, j \in \mathcal{I}'_N, j \neq s, \tag{9}$$

$$t'_{sj} = \frac{t_{rj}}{t_{rs}} \; j \in \mathcal{I}'_N, j \neq r, \tag{10}$$

$$t'_{ir} = -\frac{t_{is}}{t_{rs}} \; j \in \mathcal{I}'_B, j \neq s, \tag{11}$$

$$t'_{sr} = \frac{1}{t_{rs}}, \tag{12}$$

ahol $\mathcal{I}'_B = \mathcal{I}_B - \{r\} \cup \{s\}$ *az új generáló rendszer indexhalmaza és* t'_{ij} *az előállításában szereplő új együtthatók.*

2.3. Tétel (Kicserélési tétel). *[15] Legyen* $\{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_n\}$ *a* V *vektortér bázisa. Ha* $\{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_k\}$ *lineárisan független vektorrendszer, akkor a* $\{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_n\}$ *bázisból alkalmas* k *számú vektort elhagyva és azokat rendre az* $\{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_k\}$ *vektorokkal helyettesítve ismét a tér bázisát nyerjük.*

A kicserélési tétel nyilvánvaló következménye a következő tétel.

2.4. Tétel. *[15] Ha a* V *vektortér* n *dimenziós, akkor bármely* $n+1$ *vektorra lineárisan összefüggő. Ha* $n \geq 1$*, akkor bármely* n *lineárisan független vektor a* V *tér bázisát alkotja.*

2.1. Lineáris egyenletrendszerek

2.7. Definíció (Megoldhatóság). *Legyen adott az* $A \in \mathbb{R}^{m \times n}$ *és a* $\mathbf{b} \in \mathbb{R}^m$ *vektor. Az*

$$A\mathbf{x} = \mathbf{b}$$

egyenletet megoldható*nak nevezzük, ha létezik* $\mathbf{s} \in \mathbb{R}^n$ *vektor, amelyre*

$$\mathbf{s}_1\mathbf{a}_1 + \mathbf{s}_2\mathbf{a}_2 + ... + \mathbf{s}_n\mathbf{a}_n = \mathbf{b}$$

teljesül, ahol az $\mathbf{a}_i$ *az* A *mátrix* i*. oszlop vektora. Az* $\mathbf{x} = \mathbf{s}$ *vektort az egyenletrendszer megoldásának nevezzük.*

2.1. Lemma (Rouché-Kronecker-Capelli lemma). *Az alábbi két lineáris egyenletrendszer közül pontosan az egyik oldható meg:*

$$A\mathbf{x} = \mathbf{b} \quad \} \quad (\epsilon_1) \qquad\qquad \left.\begin{array}{c} y^T A = 0 \\ y^T b = 1 \end{array}\right\} \quad (\epsilon_2)$$

2.5. Tétel. *Az $Ax = b$ egyenletrendszer megoldhatóságát az alábbi módon is jellemezhetjük:*
Legyen $A \in \mathbb{R}^{m\times n}$ és $\mathbf{b} \in \mathbb{R}^m$. Ekkor az alábbi állítások ekvivalensek:

1. *Az $A\mathbf{x} = \mathbf{b}$ megoldható.*
2. *A $\mathbf{b} \in \mathcal{L}\{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n\}$, ahol $\mathbf{a}_i$ az A mátrix i. oszlopvektora.*
3. *$rang(\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n) = rang(\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n, \mathbf{b})$.*

Bizonyítás.
1. $\Rightarrow$ 2. Ha $A\mathbf{x} = \mathbf{b}$ megoldható akkor $\exists\lambda$, hogy $\sum_{i=1}^{n} \lambda_i \mathbf{a}_i = \mathbf{b}$ $(A\lambda = \mathbf{b})$ így $\mathbf{b} \in \mathcal{L}\{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n\} = \{\mathbf{b} : \exists\lambda_i \in \mathbb{R} : \sum_{i=1}^{n} \lambda_i \mathbf{a}_i = \mathbf{b}\}$.
2. $\Rightarrow$ 3. $rang(\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n) \leq rang(\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n, \mathbf{b})$ triviális a definíció miatt. Legyen B bázisa $\{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n\}$-nek. Ha $\mathbf{b} \in \mathcal{L}\{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n\}$, vagyis $\exists\lambda_i \in \mathbb{R}$, hogy $\mathbf{b} = \sum_{i=1}^{n} \lambda_i \mathbf{a}_i$, akkor B generálja a $\{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n, \mathbf{b}\}$-t, így $rang(\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n) = rang(\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n, \mathbf{b})$
3. $\Rightarrow$ 1. Legyen B bázisa $\{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n\}$-nek. Mivel $rang(\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n) = rang(\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n, \mathbf{b})$, így B generálja $\{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n, \mathbf{b}\}$-t is, de $B \subseteq \{(\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n\}$. Ekkor $\exists\lambda_i$, hogy $\sum_{i=1}^{n} \lambda_i \mathbf{a}_i = \mathbf{b}$, vagyis az $A\mathbf{x} = \mathbf{b}$ megoldható. ■

2.2. Mátrixok

2.6. Tétel (Mátrix rang tétel). *Tetszőleges*

$$A = (\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n) = (\mathbf{a}^{(1)}, \mathbf{a}^{(2)}, ..., \mathbf{a}^{(m)})^T$$

mátrix esetén $rang(\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n) = rang(\mathbf{a}^{(1)}, \mathbf{a}^{(2)}, ..., \mathbf{a}^{(m)})$, ahol az A mátrix oszlopvektorai az $(\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n)$, míg sorvektorai az $(\mathbf{a}^{(1)}, \mathbf{a}^{(2)}, ..., \mathbf{a}^{(m)})$ vektorok.

Vagyis a mátrix sorvektorainak a rangja megegyezik a mátrix oszlopvektorainak a rangjával.

2.8. Definíció (Szinguláris mátrix). *Legyen az $A \in \mathbb{R}^{m\times m}$ négyzetes mátrix. Az A mátrixot* szinguláris*nak nevezzük, ha az oszlop vektorai lineárisan összefüggnek. A nem szinguláris A mátrixot* reguláris*nak nevezzük.*

Példa: A $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_4$ vektorokból álló mátrix szinguláris, mert az $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$ vektorok lineárisan összefüggnek. Az $\mathbf{a}_2$ vektor az $\mathbf{a}_1$ vektor fele és az $\mathbf{a}_3$ vektor pedig a háromszorosa az $\mathbf{a}_2$-nek, vagyis 3/2-szerese az $\mathbf{a}_1$ vektornak. Az $\mathbf{a}_4$ vektor pedig nem fejezhető ki a másik három vektor segítségével. Ezért a rang csupán csak 2.

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$
1	1/2	3/2	4
2	1	3	3
3	3/2	9/2	2
4	2	6	1

2.9. Definíció (Invertálhatóság). *Legyen az $A \in \mathbb{R}^{m\times m}$ mátrix. Az A mátrixot* invertálható*nak nevezzük, ha létezik egy $B \in \mathbb{R}^{m\times m}$ mátrix, melyre $AB = BA = I$ teljesül, ahol az $I \in \mathbb{R}^{m\times m}$ az egység mátrix. A B mátrixot az A mátrix* inverz*ének nevezzük.*

2.10. Definíció (Inverzió). *[15] Legyen $\sigma \in S_m$ (S_m az $\{1, 2, ..., m\}$ elemek permutációinak a halmaza.). Ha $i, j \in \mathbb{N}$, $i < j$ és $\sigma(i) > \sigma(j)$, akkor azt mondjuk, hogy $\sigma(i), \sigma(j)$* inverzió*t alkot. A σ permutáció inverzióinak a számát $I(\sigma)$-val jelöljük.*

2.11. Definíció (Determináns). *Legyen az $A \in \mathbb{R}^{m\times m}$ mátrix. Az A mátrix* determináns*án a*

$$det(A) = \sum_{\sigma\in\S_m} (-1)^{i(\sigma)} a_{1\sigma(1)} a_{2\sigma(2)} ... a_{m\sigma(m)}$$

számot értjük, ahol S_m az $\{1, 2, ..., m\}$ elemek permutációinak halmazát jelöli, σ egy adott permutáció és $I(\sigma)$ a σ permutáció inverzióinak a száma.

2.7. Tétel (Cramer-szabály). *Legyen $A \in \mathbb{R}^{m\times m}$ reguláris mátrix és legyen $\mathbf{b} \in \mathbb{R}^m$ tetszőleges vektor. Ekkor az $A\mathbf{x} = \mathbf{b}$ megoldható és $x_i = \frac{det(A_i)}{det(A)}$ teljesül bármely i = 1,2,...,m indexre, ahol az $A_i \in \mathbb{R}^{m\times m}$ mátrixot úgy nyertük az A mátrixból, hogy az i. oszlopát, az $\mathbf{a}_i$ vektort kicseréltük a $\mathbf{b}$ vektorral.*

2.8. Tétel. *Legyen $A \in \mathbb{R}^{m\times m}$. A következő állítások ekvivalensek:*

1. *Az A mátrix reguláris.*
2. *Az A mátrix invertálható.*
3. *$det(A) \neq 0$.*

4. Az $A\mathbf{x} = \mathbf{b}$ *egyenletrendszer bármely* $\mathbf{b} \in \mathbb{R}^m$ *esetén megoldható.*

Bizonyítás.
1. $\Rightarrow$ 2. Mivel A oszlopai lineárisan függetlenek, így kifeszítik az $\mathbb{R}^m$ teret, és így $A\mathbf{x} = \mathbf{b}$ minden $\mathbf{b}$-re megoldható. Oldjuk meg szimultán az $[A|I]$ rendszert. Ez megtehető, mert A rangja m, és eredményül az $[I|A^{-1}]$-et kapjuk.
2 $\Rightarrow$ 3. $1 = det(I) = det(AA^{-1}) = det(A)det(A^{-1}) \Rightarrow det(A) \neq 0$
3. $\Rightarrow$ 4. A Cramer-szabály miatt $\mathbf{x}$ meghatározható.
4. $\Rightarrow$ 1. Indirekten tegyük fel, hogy A oszlopai összefüggnek. Ekkor $rang(A) < m$ lenne, és akkor létezik olyan $\mathbf{b} \in \mathbb{R}^m$ hogy, $\mathbf{b} \notin \mathcal{L}\{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_m\}$ lineáris térnek. Vagyis az $A\mathbf{x} = \mathbf{b}$ nem megoldható. ■

Egy vektorrendszer és egy mátrix bázisának meghatározásához a Gauss-Jordan eliminációs algoritmust használjuk.

2.3. Gauss-Jordan elimináció

bemenő adatok: $m, n \in \mathbb{N}, \mathcal{J} = \{1, 2, ..., n\}$ indexhalmaz, $A \in \mathbb{R}^{m \times n}$ és $\mathbf{b} \in \mathbb{R}^m$;
begin
  $k := 1$, és $\mathcal{J}_B = \emptyset$;
  while$(k \leq m)$**do**
    if$(\mathbf{a}_{kj} = 0\ \forall j \in \mathcal{J}$ és $\mathbf{b}_k = 0)$
    then
      a pivot tábla k. sorát töröljük;
      legyen m := m - 1;
    else
      if$(\mathbf{a}_{kj} = 0\ \forall j \in \mathcal{J}$ és $\mathbf{b}_k \neq 0)$
      then
        STOP: nem létezik $\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}$;
      else
        $\exists l \in \mathcal{J} : \mathbf{a}_{kl} \neq 0$;
        pivotáljunk a (k,l) elemen és $\mathcal{J}_B := \mathcal{J}_B \cup \{l\}$;
        k := k + 1;
      endif
    endif
  endwhile
  STOP: $\exists \mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}$;
end

Legyen $A \in \mathbb{R}^{m\times n}$ és $\mathbf{b} \in \mathbb{R}$. Az $A\mathbf{x} = \mathbf{b}$ egyenletrendszert a Gauss-Jordan eliminációs eljárással $\mathrm{O}(m)$ iterációban és $\mathrm{O}(m^2 n)$ aritmetikai művelet alkalmazásával lehet megoldani.

Példa: A Gauss-Jordan elimináció segítségével belátjuk, hogy a

$$B = \begin{array}{|cccccc|}\hline 1 & 2 & 2 & 0 & 7 & 0 \\ 1 & 3 & 0 & 0 & 5 & 0 \\ 0 & 4 & 3 & 1 & -1 & 2 \\ 0 & 0 & 0 & 0 & 2 & 4 \\ \hline \end{array}$$
$$\begin{array}{cccccc} \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 & \mathbf{a}_4 & \mathbf{a}_5 & \mathbf{a}_6 \end{array}$$

vektorrendszer rangja 4, és egy bázisa az $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_4, \mathbf{a}_6$ vektorokból áll.

	$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{e}_1$	$\mathbf{e}_2$	$\mathbf{e}_3$	$\mathbf{e}_4$
$\mathbf{e}_1$	1	2	2	0	7	0	1	0	0	0
$\mathbf{e}_2$	1	3	0	0	5	0	0	1	0	0
$\mathbf{e}_3$	0	4	3	**1**	-1	2	0	0	1	0
$\mathbf{e}_4$	0	0	0	0	2	4	0	0	0	1

Mindig azon az elemen fogunk báziscserét csinálni ami be van keretezve. Most a 3. sor 4. elemén fogunk. A továbbiakban jelölje $t^{(i)}$ az i-edik sort.

	$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{e}_1$	$\mathbf{e}_2$	$\mathbf{e}_3$	$\mathbf{e}_4$
$\mathbf{e}_1$	1	2	2	0	7	0	1	0	0	0
$\mathbf{e}_2$	**1**	3	0	0	5	0	0	1	0	0
$\mathbf{a}_4$	0	4	3	1	-1	2	0	0	1	0
$\mathbf{e}_4$	0	0	0	0	2	4	0	0	0	1

A második sor első elemén elvégzett báziscsere alkalmával a következő transzformációt kellett elvégezni: $t^{(1)} = t^{(1)} - t^{(2)}$.

	$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{e}_1$	$\mathbf{e}_2$	$\mathbf{e}_3$	$\mathbf{e}_4$
$\mathbf{e}_1$	0	-1	2	0	2	0	1	-1	0	0
$\mathbf{a}_1$	1	3	0	0	5	0	0	1	0	0
$\mathbf{a}_4$	0	4	3	1	-1	2	0	0	1	0
$\mathbf{e}_4$	0	0	0	0	2	**4**	0	0	0	1

Az elvégzett transzformációk: $t^{(4)} = \frac{1}{4}t^{(4)}$, $t^{(3)} = t^{(3)} - 2t^{(4)}$.

	$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{e}_1$	$\mathbf{e}_2$	$\mathbf{e}_3$	$\mathbf{e}_4$
$\mathbf{e}_1$	0	**-1**	2	0	2	0	1	-1	0	0
$\mathbf{a}_1$	1	3	0	0	5	0	0	1	0	0
$\mathbf{a}_4$	0	4	3	1	-2	0	0	0	1	-1/2
$\mathbf{a}_6$	0	0	0	0	1/2	1	0	0	0	1/4

Az elvégzett transzformációk: $t^{(1)} = -1t^{(1)}$, $t^{(2)} = t^{(2)} - 3t^{(1)}$ és $t^{(3)} = t^{(3)} - 4t^{(1)}$.

	$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{e}_1$	$\mathbf{e}_2$	$\mathbf{e}_3$	$\mathbf{e}_4$
$\mathbf{a}_2$	0	1	-2	0	-2	0	-1	1	0	0
$\mathbf{a}_1$	1	0	6	0	11	0	3	-2	0	0
$\mathbf{a}_4$	0	0	11	1	6	0	4	-4	1	-1/2
$\mathbf{a}_6$	0	0	0	0	1/2	1	0	0	0	1/4

Az utolsó táblából jól látszik, hogy egy B bázis tényleg az $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_4, \mathbf{a}_6$ vektorokból áll, és megjelent a mátrix jobb oldalán a bázis inverze is:

$$B^{-1} = \begin{array}{|cccc|}\hline \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 & \mathbf{e}_4 \\ \hline -1 & 1 & 0 & 0 \\ 3 & -2 & 0 & 0 \\ 4 & -4 & 1 & 1/2 \\ 0 & 0 & 0 & 1/4 \\ \hline \end{array}$$

Ellenőrzés: Ahhoz, hogy belássuk, hogy ez tényleg a B-nek az inverze, össze kell szorozni a bázisvektorokból álló mátrixot az inverz mátrixszal, majd egy egység mátrixot kell hogy kapjunk. A vektoroknak a bázisban nem az index szerinti sorrendje, hanem a bázisban elfoglalt pozíciója jelenik meg.

				-1	1	0	0
				3	-2	0	0
				4	-4	1	-1/2
				0	0	0	1/4
2	1	0	0	1	0	0	0
3	1	0	0	0	1	0	0
4	0	1	2	0	0	1	0
0	0	0	4	0	0	0	1

2.12. Definíció (Bázis megoldás). *Legyen $A \in \mathbb{R}^{m\times n}$ mátrix, $\mathbf{b} \in \mathbb{R}^m$ vektor adott és tegyük fel, hogy $rang(A) = m$, és legyen A_B az A egy bázisa. Az $\mathbf{x}_B = A_B^{-1}\mathbf{b}, \mathbf{x}_N = 0$ megoldást az $A\mathbf{x} = \mathbf{b}$ lineáris egyenletrendszer* bázis megoldás*ának nevezzük.*

2.13. Definíció (Konvex halmaz). $U \in \mathbb{R}^n$ konvex*, ha $\forall \mathbf{u}, \mathbf{v} \in U$ és $\alpha \in [0, 1]$ esetén $\alpha\mathbf{u} + (1 - \alpha)\mathbf{v} \in U$.*

2.14. Definíció (Poliéder). *Legyen $\mathcal{P} \subset \mathbb{R}^n$. A $\mathcal{P}$ halmazt* (konvex) poliéder*nek nevezzük, ha felírható*

$$\mathcal{P} = \{\mathbf{x} \in \mathbb{R}^n | A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0\}$$

alakban.

2.15. Definíció (Megengedettségi feladat, megengedett megoldás). *Tekintsük a következő lineáris egyenlőtlenség rendszert*

$$\begin{aligned} A\mathbf{x} &= \mathbf{b} \\ \mathbf{x} &\geq 0 \end{aligned} \tag{13}$$

ahol $A \in \mathbb{R}^{m \times n}, \mathbf{b} \in \mathbb{R}^m, \mathbf{x} \in \mathbb{R}^n$. A (13) lineáris egyenlőtlenség rendszert megengedettségi feladat*nak nevezzük. A*

$$\mathcal{P} := \{\mathbf{x} \in \mathbb{R}^n | A\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq 0\}$$

halmazt megengedett megoldás halmaz*nak nevezzük. Nyilván, a $\mathcal{P}$ halmaz poliéder. $\mathcal{P} = \emptyset$ is előfordulhat.*

Feltehetjük hogy rang(A) = m, ui. a rang ellenőrzése, és a redundáns feltételek elhagyása a Gauss-Jordan elimináció segítségével egyszerre megtehető.

Példa: A következőkben adott lineáris egyenletrendszer és jobboldal közül a Gauss-Jordan elimináció segítségével elhagyjuk a redundáns feltételeket, így kapunk egy bázist és a mátrix rangját.

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{b}$
1	2	0	4	3	5	1
3	4	1	0	0	0	2
3	6	0	12	9	15	3
2	4	0	0	0	5	4
0	1	0	0	1	0	2
6	8	2	0	0	0	4

Kezdőbázisnak hozzávesszük az egység mátrixot a jobb oldalra. Ennek most a számolásnál nincs nagy szerepe, ezért később el is hagyjuk. A báziscserét ismét mindig a bekeretezett elemen végezzük.

	$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{e}_1$	$\mathbf{e}_2$	$\mathbf{e}_3$	$\mathbf{e}_4$	$\mathbf{e}_5$	$\mathbf{e}_6$	$\mathbf{b}$
$\mathbf{e}_1$	1	2	0	4	3	5	1	0	0	0	0	0	1
$\mathbf{e}_2$	3	4	1	0	0	0	0	1	0	0	0	0	2
$\mathbf{e}_3$	3	6	0	12	9	15	0	0	1	0	0	0	3
$\mathbf{e}_4$	2	4	0	0	0	5	0	0	0	1	0	0	4
$\mathbf{e}_5$	0	1	0	0	1	0	0	0	0	0	1	0	2
$\mathbf{e}_6$	6	8	2	0	0	0	0	0	0	0	0	1	4

Az elvégzett transzformáció a következő: $t^{(6)} = t^{(6)} - 2t^{(2)}$.

	$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{e}_1$	$\mathbf{e}_2$	$\mathbf{e}_3$	$\mathbf{e}_4$	$\mathbf{e}_5$	$\mathbf{e}_6$	$\mathbf{b}$
$\mathbf{e}_1$	1	2	0	4	3	5	1	0	0	0	0	0	1
$\mathbf{a}_3$	3	4	1	0	0	0	0	1	0	0	0	0	2
$\mathbf{e}_3$	3	6	0	12	9	15	0	0	1	0	0	0	3
$\mathbf{e}_4$	2	4	0	0	0	5	0	0	0	1	0	0	4
$\mathbf{e}_5$	0	1	0	0	1	0	0	0	0	0	1	0	2
$\mathbf{e}_6$	0	0	0	0	0	0	0	-2	0	0	0	1	0

Most elhagyjuk a jobb oldalról az egységmátrixot, és elvégezzük a pivotálást az 1. sor 4. elemén.
Az elvégzett transzformációk: $t^{(1)} = \frac{1}{4}t^{(1)}$ és $t^{(3)} = t^{(3)} - 12t^{(1)}$.

	$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{b}$
$\mathbf{a}_4$	1/4	1/2	0	1	3/4	5/4	1/4
$\mathbf{a}_3$	3	4	1	0	0	0	2
$\mathbf{e}_3$	0	0	0	0	0	0	0
$\mathbf{e}_4$	2	4	0	0	0	5	4
$\mathbf{e}_5$	0	1	0	0	1	0	2
$\mathbf{e}_6$	0	0	0	0	0	0	0

Látszik, hogy az $\mathbf{e}_3$ és $\mathbf{e}_6$ sora csupa 0 elemet tartalmaz. Ezek azok a sorok, amelyeket el lehet hagyni a későbbiekben is a megoldások során. Viszont ha egy ilyen sorban a jobb oldalon a $\mathbf{b}$ oszlopában nem nulla szám állna, akkor az azt jelentené, hogy nem létezik megoldása a lineáris egyenletrendszernek. Most elhagyjuk a fenti két bekeretezett sort.

	$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{b}$
$\mathbf{a}_4$	1/4	1/2	0	1	3/4	5/4	1/4
$\mathbf{a}_3$	3	4	1	0	0	0	2
$\mathbf{e}_4$	2	4	0	0	0	5	4
$\mathbf{e}_5$	0	1	0	0	1	0	2

Az elvégzett transzformáció: $t^{(1)} = t^{(1)} - \frac{3}{4}t^{(4)}$.

	$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{b}$
$\mathbf{a}_4$	1/4	-1/4	0	1	0	5/4	-5/4
$\mathbf{a}_3$	3	4	1	0	0	0	2
$\mathbf{e}_4$	2	4	0	0	0	5	4
$\mathbf{a}_5$	0	1	0	0	1	0	2

Az elvégzett transzformáció: $t^{(3)} = \frac{1}{5}t^{(3)}$ és $t^{(1)} = t^{(1)} - \frac{5}{4}t^{(3)}$.

	$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$	$\mathbf{b}$
$\mathbf{a}_4$	-1/4	-5/4	0	1	0	0	-9/4
$\mathbf{a}_3$	3	4	1	0	0	0	2
$\mathbf{a}_6$	2/5	4/5	0	0	0	1	4/5
$\mathbf{a}_5$	0	1	0	0	1	0	2

Most a bázist az $\mathbf{a}_4, \mathbf{a}_3, \mathbf{a}_6, \mathbf{a}_5$ vektorok alkotják, a mátrix rangja pedig 4, mivel 2 lineárisan összefüggő sort hagytunk el.

2.16. Definíció (Bázis indexhalmazok). *A következő jelöléseket fogjuk használni:*

$$\begin{aligned} I_B, I_N \subset I &= \{1,2,...,n\}, & I_B &= I_B^+ \cup I_B^0 \cup I_B^-, \text{ ahol} \\ I_B^+ &= \{i \in I_B : \mathbf{x}_i > 0\}, & I_B^0 &= \{i \in I_B : \mathbf{x}_i = 0\}, \\ I_B^- &= \{i \in I_B : \mathbf{x}_i < 0\}, & I_B^\oplus &= I_B^+ \cup I_B^0 \end{aligned}$$

Az aktuális bázis megengedett bázis, ha $I_B^- = \emptyset$.

Ha a következő fejezetben részletezett LP algoritmusok végrehajtása során, egy adott elemen való pivotálás után olyan táblához jutunk, ahol a **b** jobboldal egyik eleme negatívvá válik, és az ebben a sorban levő összes elem pozitív vagy 0, akkor az algoritmus nem folytatható tovább, mert előállt az úgynevezett primál nem megengedettségi kritérium, mely a feladat nem megengedettségét mutatja. Ebben az esetben a táblából kiolvasható a Farkas lemma szerinti bizonyítéka a nem megengedettségnek.

2.17. Definíció (Primál nem megengedettségi kritérium). *Ha* $\exists \mathbf{r} \in I_B^- : J_r^- = \emptyset$ *akkor* $\mathcal{P} = \emptyset$, *ahol* $J_r^- = \{i \in I_N : t_{ri} < 0\}$ *és* $J_r^\oplus = \{i \in I_N : t_{ri} \geq 0\}$

2.18. Definíció (Döntési paraméterek). $\theta_1 := \frac{b_r}{t_{rs}} > 0$, *és*
$\theta_2 := min\{\frac{\overline{b_k}}{t_{ks}} | k \in \mathcal{I}_B^\oplus, t_{ks} > 0\} = \frac{\overline{b_q}}{t_{qs}} \geq 0$

Az itt definiált θ_1 és θ_2 változót majd az MBU-Szimplex algoritmusnál fogjuk felhasználni.

2.19. Definíció (Degeneráltság). *Egy bázist* degenerált*nak nevezünk, ha a hozzátartozó bázis megoldásnak létezik* 0 *komponense. Ez azt jelenti, hogy a* $\mathbf{b}$ *vektor előáll kevesebb, mint rangszámnyi A-beli oszlop lineáris kombinációjaként.*
Egy feladat degenerált*, ha létezik degenerált bázisa.*

Az látható, hogy ha egy bázis nem degenerált, akkor a később ismertetett algoritmusok közül, mind a Szimplex, mind az MBU algoritmus esetén javul a célfüggvény értéke, és mivel a célfüggvény monoton változik, ezért az adott bázis többet nem térhet vissza, így az eljárás véges.
Általános esetben azonban előfordulhat, (vagyis, hogyha az index választásnál a lehetséges halmazból tetszőlegesen választunk) hogy mind a Szimplex, mind az MBU algoritmus ciklizálhat. Ugyanez igaz a Criss-Cross algoritmusra is, de ott az algoritmust rögtön a minimálindexes szabály figyelembevételével fogalmaztuk meg.
Mivel a lehetséges bázisok száma véges $\left(\leq \binom{n}{m} \right)$, így az algoritmus ha nem véges, az csak úgy lehetséges, hogy ha ciklizál. Ennek elkerülését ún. indexválasztási szabályok alkalmazásával lehet biztosítani. Ilyen szabályok például a lexikografikus szabály, a minimálindex szabály (Bland-szabály), a most often selected variable és a lifo [36]. Ezek bármelyikének a használata esetén mind a Szimplex, mind az MBU algoritmus tetszőleges feladat esetén véges [5], [17], [18].

3. Algoritmusok

A következőkben bemutatunk három algoritmust és mindegyik után példaként ott fog szerepelni egy könnyebb és egy nehezebb példa megoldása is.

3.1. Definíció (Primál Lineáris Programozási feladat). *A*

$$\begin{aligned} &\min \mathbf{c}^T\mathbf{x} \\ &A\mathbf{x} = \mathbf{b} \\ &\mathbf{x} \geq 0, \end{aligned} \tag{P}$$

feladatot, ahol $A \in \mathbb{R}^{m\times n}, \mathbf{b} \in \mathbb{R}^m, \mathbf{x} \in \mathbb{R}^n$*, lineáris programozási feladatnak (LP) nevezzük.*

3.2. Definíció (Duál Lineáris Programozási feladat).

$$\begin{aligned} &\max \mathbf{b}^T\mathbf{y} \\ &A^T\mathbf{y} \leq c \end{aligned} \tag{D}$$

3.1. Tétel (Gyenge dualitás tétel). *Ha* $\mathbf{x}$ *a (P) feladat,* $\mathbf{y}$ *a (D) feladat megengedett megoldása, akkor* $\mathbf{c}^T\mathbf{x} \leq \mathbf{b}^T\mathbf{y}$

3.2. Tétel (Erős dualitás tétel). *Ha a primál feladatnak van optimális megoldása, akkor a duál feladatnak is van optimális megoldása, és a két optimális célfüggvényérték egyenlő.*

3.3. Tétel (Farkas Lemma). *Az alábbi két lineáris egyenletrendszer közül pontosan az egyik oldható meg:*

$$\left.\begin{aligned} A\mathbf{x} &= \mathbf{b} \\ \mathbf{x} &\geq 0 \end{aligned}\right\} \qquad \left.\begin{aligned} \mathbf{y}^T A &\leq 0 \\ \mathbf{y}^T\mathbf{b} &= 1 \end{aligned}\right\}$$

A primál és duál feladatok viszonyára a következő szabályokat foglalhatjuk össze:

Primál/Duál	$\nexists$ megeng. mego.	$\exists$ opt. mego	Nem korl. feladat
$\nexists$ megeng. mego.	$\surd$	x	$\surd$
$\exists$ opt. mego	x	$\surd$(Dualitás)	x
Nem korl. feladat	$\surd$	x	x(Gyenge dualitás)

Ahol a táblázatban a pipa azt jelenti, hogy van ilyen, míg az x azt, hogy nem létezik ilyen feladat.
A Gyenge dualitás tétel miatt tudjuk, hogy nem létezik olyan feladat amelynél a primál feladat, és a duál feladat sem korlátos. Mert, a tétel miatt, ha bármelyiknek létezne megoldása, akkor annak célfüggvény értéke korlát lenne a másikéra, így az csak korlátos lehet.
A következő feladat pedig példa arra az esetre, amikor a primál és a duál feladatnak egyaránt nem létezik megoldása:

$$\begin{aligned} \max(2\mathbf{x}_1 + \mathbf{x}_2) \\ \mathbf{x}_1 + \mathbf{x}_2 = 2 \\ \mathbf{x}_1 + \mathbf{x}_2 = 1 \end{aligned}$$

Ezt átalakítva a következő formára

$$\begin{aligned} \max(2\mathbf{x}_1^+ + 2\mathbf{x}_1^- + \mathbf{x}_2^+ + \mathbf{x}_2^-) \\ \mathbf{x}_1^+ - \mathbf{x}_1^- + \mathbf{x}_2^+ - \mathbf{x}_2^- \leq \ \ 2 \\ -\mathbf{x}_1^+ + \mathbf{x}_1^- - \mathbf{x}_2^+ + \mathbf{x}_2^- \leq -2 \\ \mathbf{x}_1^+ - \mathbf{x}_1^- + \mathbf{x}_2^+ - \mathbf{x}_2^- \leq \ \ 1 \\ -\mathbf{x}_1^+ + \mathbf{x}_1^- - \mathbf{x}_2^+ + \mathbf{x}_2^- \leq -1 \end{aligned}$$

könnyen látható, hogy ennek a duálja önmaga. Tehát egyiknek sem létezik megengedett megoldása.

A lineáris programozási feladatoknak sokféle alakja létezik, amelyek egymással ekvivalensek, de mi a fenti, ún. kanonikus alakot fogjuk használni.

A következő algoritmusoknál mindig a fenti LP feladat megoldására törekszünk.

A következőkben minden algoritmus után megoldjuk az alábbi két mintafeladatot:

$$\begin{aligned} \max(\mathbf{x}_1 + 2\mathbf{x}_2 + 4\mathbf{x}_3 + \mathbf{x}_4) \\ \mathbf{x}_1 + \mathbf{x}_3 + \mathbf{x}_4 \leq 5 \\ \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_4 \leq 6 \\ \mathbf{x}_3 \leq 7 \\ \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4 \geq 0 \end{aligned}$$

$$\begin{aligned} \max(\mathbf{x}_1 + 2\mathbf{x}_2 + 3\mathbf{x}_3 + 2\mathbf{x}_4 + 3\mathbf{x}_5) \\ \mathbf{x}_1 + 2\mathbf{x}_2 + \mathbf{x}_4 + \mathbf{x}_5 &\leq 110 \\ \mathbf{x}_1 + 2\mathbf{x}_3 + \mathbf{x}_4 + \mathbf{x}_5 &= 80 \\ 2\mathbf{x}_2 + \mathbf{x}_5 &\geq 40 \\ \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5 &\geq 0 \end{aligned}$$

Mivel ezek maximalizáló feladatok, de a mi algoritmusaink minimalizálóak, ezért a feladatokat át kell alakítani minimalizáló alakra:

$$\begin{aligned} \min(-\mathbf{x}_1 - 2\mathbf{x}_2 - 4\mathbf{x}_3 - \mathbf{x}_4) \\ \mathbf{x}_1 + \mathbf{x}_3 + \mathbf{x}_4 &\leq 5 \\ \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_4 &\leq 6 \\ \mathbf{x}_3 &\leq 7 \\ \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4 &\geq 0 \end{aligned}$$

$$\begin{aligned} \min(-\mathbf{x}_1 - 2\mathbf{x}_2 - 3\mathbf{x}_3 - 2\mathbf{x}_4 - 3\mathbf{x}_5) \\ \mathbf{x}_1 + 2\mathbf{x}_2 + \mathbf{x}_4 + \mathbf{x}_5 &\leq 110 \\ \mathbf{x}_1 + 2\mathbf{x}_3 + \mathbf{x}_4 + \mathbf{x}_5 &= 80 \\ 2\mathbf{x}_2 + \mathbf{x}_5 &\geq 40 \\ \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5 &\geq 0 \end{aligned}$$

Az algoritmusok során kapott pozitív megoldás az eredeti maximalizáló feladat eredményét adja majd meg.

Általánosságban a következőket állíthatjuk. A feladat a következő alakban adott:

$$\begin{aligned} A_1\mathbf{x} &\leq \mathbf{b}_1 \\ A_2\mathbf{x} &= \mathbf{b}_2 \\ \mathbf{x} &\geq 0 \end{aligned}$$

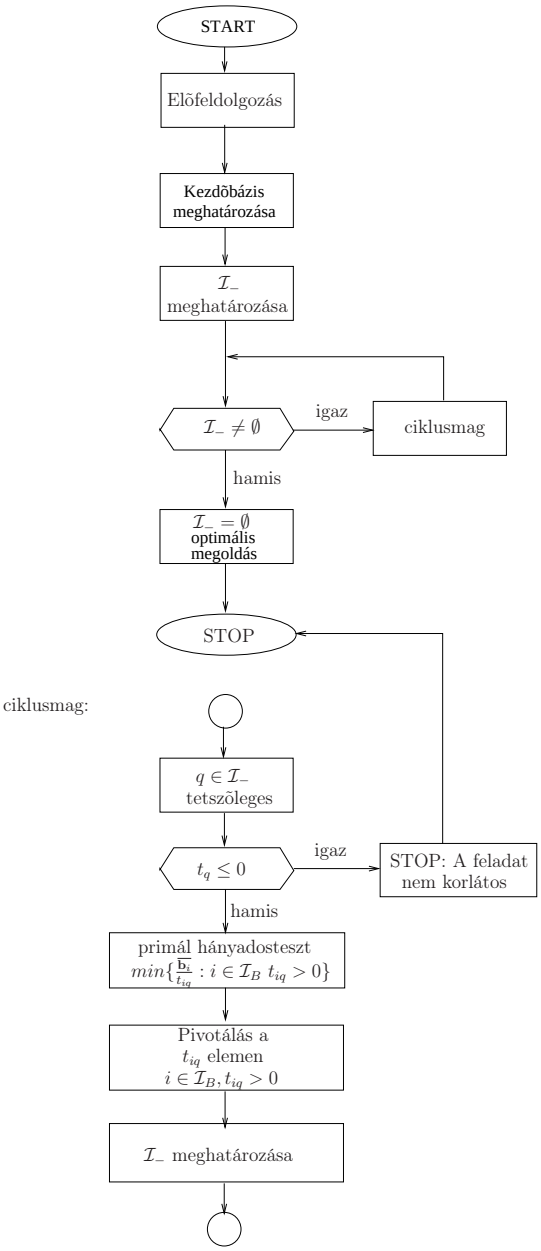
Itt feltehető, hogy $\mathbf{b}_1, \mathbf{b}_2 \geq 0$, és minden feladat ilyen alakra hozható. Ezt a feladatot átalakíthatjuk slack változók bevezetésével a következő kanonikus alakra:

$$\begin{aligned} A_1\mathbf{x} + \mathbf{u}_1 &= \mathbf{b}_1 \\ A_2\mathbf{x} + \mathbf{u}_2^* &= \mathbf{b}_2 \\ \mathbf{x}, \mathbf{u}_1, \mathbf{u}_2^* &\geq 0 \end{aligned}$$

Akkor lesz az eredeti feladatnak egy megengedett bázis megoldása, ha létezik itt egy olyan megengedett bázis megoldás, melyben $\mathbf{u}_2^* = 0$. Az ilyen bázismegoldás előállítását nevezzük első fázisnak. Az első fázisban az $\mathbf{u}_2^* = 0$ cél elérését az úgynevezett másodlagos célfüggvény segíti.
Legegyszerűbb a $\min \sum \mathbf{u}_2^*$ célfüggvényt használni, azonban $\mathbf{u}_2^*$ benne van a kezdeti bázisban, így ezt a célfüggvényt előbb transzformálni kell. Így a $\max \mathbf{1}^T A_2\mathbf{x}$ célfüggvényt kapjuk. Az első fázis feladata mindig korlátos (Látszik az $A_2\mathbf{x} \leq \mathbf{b}$ feltételből.), és természetesen van megengedett megoldása, az induló $\mathbf{u}_1, \mathbf{u}_2^*$.
Amennyiben minimalizáló célfüggvény esetén az optimum nem nulla (maximalizáló célfüggvény esetén nem $\mathbf{1}^T\mathbf{b}_2$), az eredeti feladatnak nem létezik megengedett megoldása. Ebben az esetben persze az $\mathbf{u}_2^*$-os változók nullák.
Előfordulhat az, hogy minden $\mathbf{u}_2^* = 0$, de egyes indexei bázisban vannak (0 a jobboldal). Ilyenkor ezeket degenerált báziscserékkel kivisszük a bázisból.
Ha már *-os változók nincsenek a bázisban, akkor a hozzájuk tartozó oszlopokat elhagyjuk a feladatból és az ún. második fázis során az eredeti célfüggvénnyel számolunk.

3.1. Primál Szimplex módszer

bemenő adatok:
a kanonikus (P) feladat A_B primál megengedett bázisához tartozó T_B (rövid) bázis tábla;
begin
  $\mathcal{I}_- := \{i \in \mathcal{I}_N | \mathbf{c}_i < 0\}$;
  while $(\mathcal{I}_- \neq \emptyset)$ **do**
    legyen $q \in \mathcal{I}_-$ tetszőleges;
    if$(\mathbf{t}_q \leq 0)$
    then STOP: a feladat nem korlátos;
    else
      legyen $\vartheta := min\{\frac{\overline{b_i}}{t_{iq}} : i \in \mathcal{I}_B \; t_{iq} > 0\}$;(primál hányadosteszt)
      legyen $p \in \mathcal{I}_B$ tetszőleges, melyre $\frac{\overline{b_p}}{t_{pq}} = \vartheta$;
    endif
    pivotálás: $\mathcal{I}_B := \mathcal{I}_B \cup \{q\} \setminus \{p\}$;
    az $\mathcal{I}_-$ indexhalmaz meghatározása;
  endwhile
  $\mathcal{I}_- = \emptyset$, optimális megoldásnál vagyunk;
end



1. ábra. Szimplex módszer.

Tegyük fel, hogy a megoldandó feladatunk $A\mathbf{x} = \mathbf{b}$, $\mathbf{x}, \mathbf{b} \geq \mathbf{0}$ alakú. Ha kezdetben nem ilyen a feladat, akkor ún. slack változók bevezetésével (hozzáadásával, kivonásával) ilyen alakra hozzuk. A megengedett megoldás az alábbi feladat megoldásával kereshető:

$$\begin{aligned} \max\ & \mathbf{1}^T A\mathbf{x}, \\ A\mathbf{x} + I\mathbf{u} &= \mathbf{b}, \\ \mathbf{x}, \mathbf{u} &\geq 0, \end{aligned}$$

ahol $\mathbf{u}$ jelöli a bevezetett mesterséges slack változókat. Ennek a feladatnak létezik optimális megoldása, ugyanis $(\mathbf{1}^T A)\mathbf{x} \leq \mathbf{1}^T\mathbf{b}$.
A $\min(\mathbf{1}^T A)\mathbf{x} = \mathbf{1}^T\mathbf{b}$ akkor és csak akkor, ha létezik olyan $\mathbf{x}$ vektor, hogy $A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}$. Ekkor a beágyazott feladatnak az $(\overline{\mathbf{x}}, \overline{\mathbf{u}})$ optimális megoldása, akkor és csak akkor ha $\overline{\mathbf{u}} = \mathbf{0}$. Ilyen esetekben lépünk át a primál Szimplex módszerben a második fázisba, és ekkor elhagyjuk a slack változók oszlopát, illetve a másodlagos célfüggvény sorát.

Példa 1:
Oldjuk meg a következő feladatot:

$$\begin{aligned} \min(-\mathbf{x}_1 - 2\mathbf{x}_2 - 4\mathbf{x}_3 - \mathbf{x}_4) & \\ \mathbf{x}_1 + \mathbf{x}_3 + \mathbf{x}_4 &\leq 5 \\ \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_4 &\leq 6 \\ \mathbf{x}_3 &\leq 7 \\ \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4 &\geq 0 \end{aligned}$$

Át kell alakítanunk a fenti feladatot kanonikus LP-feladattá:

$$\begin{aligned} \min(-\mathbf{x}_1 - 2\mathbf{x}_2 - 4\mathbf{x}_3 - \mathbf{x}_4) & \\ \mathbf{x}_1 + \mathbf{x}_3 + \mathbf{x}_4 + \mathbf{u}_1 &= 5 \\ \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_4 + \mathbf{u}_2 &= 6 \\ \mathbf{x}_3 + \mathbf{u}_3 &= 7 \\ \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3 &\geq 0 \end{aligned} \tag{14}$$

Megoldás a Szimplex-algoritmussal:

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{u}_3$	$\mathbf{b}$
$\mathbf{u}_1$	1	0	1	1	1	0	0	5
$\mathbf{u}_2$	1	1	0	1	0	1	0	6
$\mathbf{u}_3$	0	0	1	0	0	0	1	7
c	-1	-2	-4	-1	0	0	0	0

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{u}_3$	$\mathbf{b}$
$\mathbf{x}_1$	1	0	1	1	1	0	0	5
$\mathbf{u}_2$	0	1	-1	0	-1	1	0	1
$\mathbf{u}_3$	0	0	1	0	0	0	1	7
c	0	-2	-3	0	1	0	0	5

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{u}_3$	$\mathbf{b}$
$\mathbf{x}_1$	1	0	1	1	1	0	0	5
$\mathbf{x}_2$	0	1	-1	0	-1	1	0	1
$\mathbf{u}_3$	0	0	1	0	0	0	1	7
c	0	0	-5	0	-1	2	0	7

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{u}_3$	$\mathbf{b}$
$\mathbf{x}_3$	1	0	1	1	1	0	0	5
$\mathbf{x}_2$	1	1	0	1	0	1	0	6
$\mathbf{u}_3$	-1	0	0	-1	-1	0	1	2
c	5	0	0	5	4	2	0	32

Tehát a keresett megoldás az $\mathbf{x}_1 = 0$, $\mathbf{x}_2 = 6$, $\mathbf{x}_3 = 5$, és $\mathbf{x}_4 = 0$. Az eredeti feladat maximális értéke 32. Ha a célfüggvény ($\mathbf{c}$) sorában pozitív szám szerepel és nincs fölötte pozitív, akkor nincs megoldása a feladatnak.

Példa 2:
Oldjuk meg a következő feladatot:

$$\begin{aligned} \min(-\mathbf{x}_1 - 2\mathbf{x}_2 - 3\mathbf{x}_3 - 2\mathbf{x}_4 - 3\mathbf{x}_5) \\ \mathbf{x}_1 + 2\mathbf{x}_2 + \mathbf{x}_4 + \mathbf{x}_5 &\leq 110 \\ \mathbf{x}_1 + 2\mathbf{x}_3 + \mathbf{x}_4 + \mathbf{x}_5 &= 80 \\ 2\mathbf{x}_2 + \mathbf{x}_5 &\geq 40 \\ \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5 &\geq 0 \end{aligned}$$

Át kell alakítanunk a fenti feladatot LP-feladattá, és be kell vezetnünk az új, másodlagos célfüggvényt is.

$$\begin{aligned} z = \min(-\mathbf{x}_1 - 2\mathbf{x}_2 - 3\mathbf{x}_3 - 2\mathbf{x}_4 - 3\mathbf{x}_5) \\ z^* = \min(\mathbf{x}_1 - 2\mathbf{x}_2 - 2\mathbf{x}_3 - \mathbf{x}_4 - 2\mathbf{x}_5 + \mathbf{u}_2) \\ \mathbf{x}_1 + 2\mathbf{x}_2 + \mathbf{x}_4 + \mathbf{x}_5 + \mathbf{u}_1 &= 110 \\ \mathbf{x}_1 + 2\mathbf{x}_3 + \mathbf{x}_4 + \mathbf{x}_5 + \mathbf{v}_2^* &= 80 \\ 2\mathbf{x}_2 + \mathbf{x}_5 - \mathbf{u}_3 + \mathbf{v}_3^* &= 40 \\ \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{u}_1, \mathbf{u}_2, \mathbf{v}_1^*, \mathbf{v}_2^* &\geq 0 \end{aligned} \tag{15}$$

Megoldás a Szimplex-algoritmussal:

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{u}_1$	$\mathbf{u}_3$	$\mathbf{v}_1^*$	$\mathbf{v}_2^*$	$\mathbf{b}$
$\mathbf{u}_1$	1	2	0	1	1	1	0	0	0	110
$\mathbf{v}_1^*$	1	0	2	1	1	0	0	1	0	80
$\mathbf{v}_2^*$	0	2	0	0	1	0	-1	0	1	40
z	-1	-2	-3	-2	-3	0	0	0	0	0
z^*	-1	-2	-2	-1	-2	0	1	0	0	0

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{u}_1$	$\mathbf{u}_3$	$\mathbf{v}_1^*$	$\mathbf{v}_2^*$	$\mathbf{b}$
$\mathbf{u}_1$	0	2	-2	0	0	1	0	-1	0	30
$\mathbf{x}_1$	1	0	2	1	1	0	0	1	0	80
$\mathbf{v}_2^*$	0	2	0	0	1	0	-1	0	1	40
z	0	-2	-1	-1	-2	0	0	1	0	80
z^*	0	-2	0	0	-1	0	1	1	0	80

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{u}_1$	$\mathbf{u}_3$	$\mathbf{v}_1^*$	$\mathbf{v}_2^*$	$\mathbf{b}$
$\mathbf{u}_1$	0	2	-2	0	0	1	0	-1	0	30
$\mathbf{x}_1$	1	-2	2	1	0	0	1	1	1	40
$\mathbf{x}_5$	0	2	0	0	1	0	-1	0	1	40
z	0	2	-1	-1	0	0	-2	1	2	160
z^*	0	0	0	0	0	0	0	1	1	120

Most a $\mathbf{v}_1^*$ és a $\mathbf{v}_2^*$ vektorok oszlopait és a másodlagos célfüggvény sorát elhagyhatjuk. Ekkor a következő táblát kapjuk:

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{u}_1$	$\mathbf{u}_3$	$\mathbf{b}$
$\mathbf{u}_1$	0	2	-2	0	0	1	0	30
$\mathbf{x}_1$	1	-2	2	1	0	0	1	40
$\mathbf{x}_5$	0	2	0	0	1	0	-1	40
z	0	2	-1	-1	0	0	-2	160

Ekkor egy megengedett megoldás lenne az $\mathbf{x}_1 = 40$, $\mathbf{x}_2 = 0$, $\mathbf{x}_3 = 0$, $\mathbf{x}_4 = 0$ és az $\mathbf{x}_5 = 40$, de mivel mi a minimumot keressük, ezért folytatjuk a pivotálást a 2. sor 3. elemén.

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{u}_1$	$\mathbf{u}_3$	$\mathbf{b}$
$\mathbf{u}_1$	1	0	0	1	0	1	1	70
$\mathbf{x}_3$	1/2	-1	1	1/2	0	0	1/2	20
$\mathbf{x}_5$	0	2	0	0	1	0	-1	40
z	1/2	1	0	-1/2	0	0	-3/2	180

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{u}_1$	$\mathbf{u}_3$	$\mathbf{b}$
$\mathbf{u}_1$	0	2	-2	0	0	1	0	30
$\mathbf{x}_4$	1	-2	2	1	0	0	1	40
$\mathbf{x}_5$	0	2	0	0	1	0	-1	40
z	1	0	1	0	0	0	-1	200

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{u}_1$	$\mathbf{u}_3$	$\mathbf{b}$
$\mathbf{u}_1$	0	2	-2	0	0	1	0	30
$\mathbf{u}_3$	1	-2	2	1	0	0	1	40
$\mathbf{x}_5$	1	0	2	1	1	0	0	80
z	2	-2	3	1	0	0	0	240

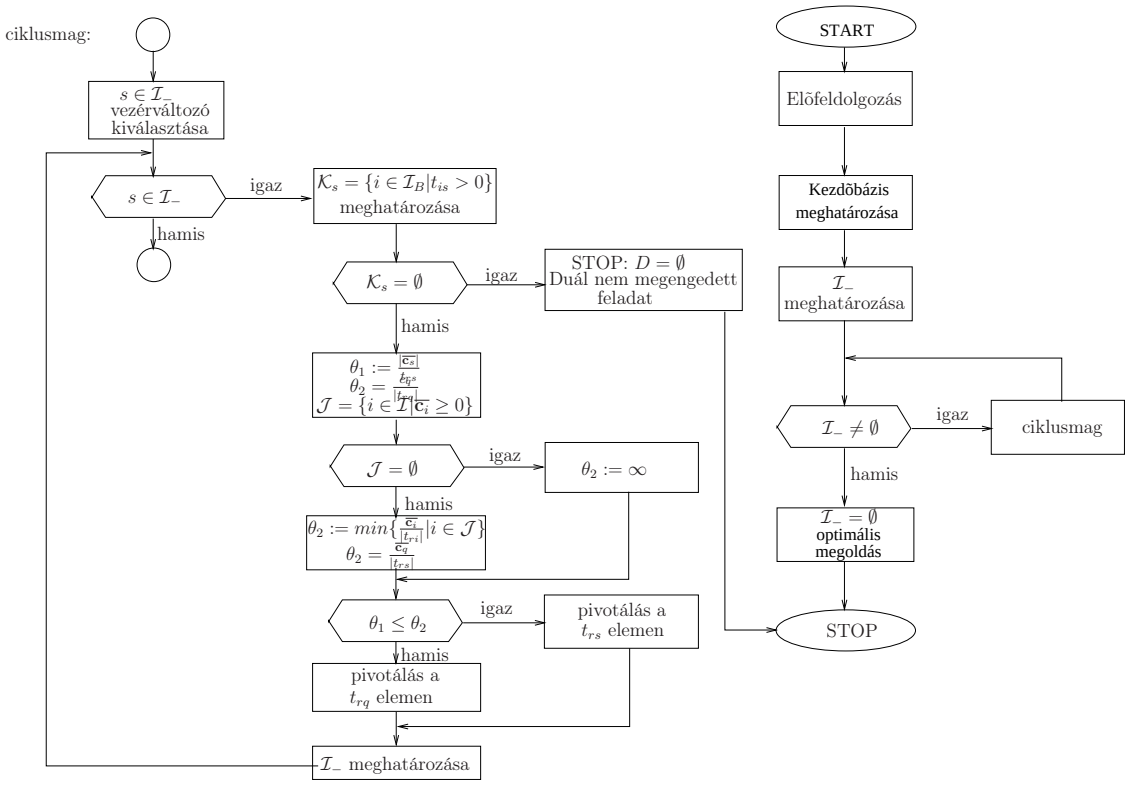
	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{u}_1$	$\mathbf{u}_3$	$\mathbf{b}$
$\mathbf{x}_2$	0	1	-1	0	0	1/2	0	15
$\mathbf{u}_3$	1	0	0	1	0	1	1	70
$\mathbf{x}_5$	1	0	2	1	1	0	0	80
z	2	0	1	1	0	1	0	270

Ezzel a maximális megoldáshoz jutottunk, ami az $\mathbf{x}_1 = 0$, $\mathbf{x}_2 = 15$, $\mathbf{x}_3 = \mathbf{x}_4 = 0$ és az $\mathbf{x}_5 = 80$ helyen vétetik fel, és az értéke 270.

3.2. MBU-Szimplex algoritmus

bemenő adatok:
a kanonikus (P) feladat A_B primál megengedett bázisához tartozó T_B (rövid) bázis tábla;
begin
$\mathcal{I}_- := \{i \in \mathcal{I}_N | \mathbf{c}_i < 0\}$;
while$(\mathcal{I}_- \neq \emptyset)$**do**
legyen $s \in \mathcal{I}_-$ tetszőleges vezető változó;
while$(s \in \mathcal{I}_-)$**do**
legyen $\mathcal{K}_s = \{i \in \mathcal{I}_B | t_{is} > 0\}$;
if$(\mathcal{K}_s = \emptyset)$
then STOP: a $D = \emptyset$;
else
legyen $\vartheta := min\{\frac{\mathbf{b}_i}{t_{is}} | i \in \mathcal{K}_s\}$;
legyen $r \in \mathcal{I}_B$ tetszőleges, melyre $\frac{\overline{\mathbf{b}_r}}{t_{rs}} = \vartheta$ és $\theta_1 := \frac{|\overline{\mathbf{c}_s}|}{t_{rs}}$;
legyen $\mathcal{J} = \{i \in \mathcal{I} | \overline{\mathbf{c}_i} \geq 0 \text{ és } t_{ri} < 0\}$, és $q \in \mathcal{I}_N : \theta_2 = \frac{c_q}{|t_{rq}|}$;
if $(\mathcal{J} = \emptyset)$
then $\theta_2 := \infty$
else $\theta_2 := min\{\frac{\overline{\mathbf{c}_i}}{|t_{ri}|} | i \in \mathcal{J}\}$, és $q \in \mathcal{I}_N : \theta_2 = \frac{\overline{\mathbf{c}_q}}{|t_{rq}|}$;

endif
if$(\theta_1 \leq \theta_2)$
then pivotálás a t_{rs} elemen;
else pivotálás a t_{rq} elemen;
endif
határozzuk meg az $\mathcal{I}_-$ halmazt;
endif
endwhile
endwhile
$\mathcal{I}_- = \emptyset$, optimális megoldásnál vagyunk;
end



2. ábra. MBU-Szimplex módszer.

Példa 1:
Oldjuk meg a következő feladatot:

$$\begin{aligned} \min(-\mathbf{x}_1 - 2\mathbf{x}_2 - 4\mathbf{x}_3 - \mathbf{x}_4) \\ \mathbf{x}_1 + \mathbf{x}_3 + \mathbf{x}_4 \leq 5 \\ \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_4 \leq 6 \\ \mathbf{x}_3 \leq 7 \\ \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4 \geq 0 \end{aligned}$$

Ahogy azt a Szimplex algoritmusnál már elvégeztük, a feladatot itt is át kell alakítanunk LP feladattá. Így most a (14)-es alakból indulunk ki.

Megoldás az MBU-Szimplex algoritmussal:

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{u}_3$	**b**
$\mathbf{u}_1$	1	0	1	1	1	0	0	5
$\mathbf{u}_2$	1	1	0	1	0	1	0	6
$\mathbf{u}_3$	0	0	1	0	0	0	1	7
c	-1	-2	-4	-1	0	0	0	0

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{u}_3$	**b**
$\mathbf{x}_1$	1	0	1	1	1	0	0	5
$\mathbf{u}_2$	0	1	-1	0	-1	1	0	1
$\mathbf{u}_3$	0	0	1	0	0	0	1	7
c	0	-2	-3	0	1	0	0	5

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{u}_3$	**b**
$\mathbf{x}_1$	1	1	0	1	0	1	0	6
$\mathbf{u}_1$	0	-1	1	0	1	-1	0	-1
$\mathbf{u}_3$	0	0	1	0	0	0	1	7
c	0	-1	-4	0	0	1	0	6

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{u}_3$	**b**
$\mathbf{x}_2$	1	1	0	1	0	1	0	6
$\mathbf{u}_1$	1	0	1	1	1	0	0	5
$\mathbf{u}_3$	0	0	1	0	0	0	1	7
c	1	0	-4	1	0	2	0	12

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{u}_3$	$\mathbf{b}$
$\mathbf{x}_2$	1	1	0	1	0	1	0	6
$\mathbf{u}_1$	1	0	1	1	1	0	0	5
$\mathbf{u}_3$	-1	0	0	-1	-1	0	1	2
c	5	0	0	5	4	2	0	32

Tehát a megoldás $\mathbf{x}_1 = 0$, $\mathbf{x}_2 = 6$, $\mathbf{x}_3 = 5$, és $\mathbf{x}_4 = 0$. Így az eredeti feladat maximális értéke 32, ahogy ezt már a Szimplex-algoritmusnál is megkaptuk.

Példa 2:
Oldjuk meg a következő feladatot:

$$\begin{aligned} \min(-\mathbf{x}_1 - 2\mathbf{x}_2 - 3\mathbf{x}_3 - 2\mathbf{x}_4 - 3\mathbf{x}_5) \\ \mathbf{x}_1 + 2\mathbf{x}_2 + \mathbf{x}_4 + \mathbf{x}_5 &\leq 110 \\ \mathbf{x}_1 + 2\mathbf{x}_3 + \mathbf{x}_4 + \mathbf{x}_5 &= 80 \\ 2\mathbf{x}_2 + \mathbf{x}_5 &\geq 40 \\ \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5 &\geq 0 \end{aligned}$$

Szintén a már átalakított (15)-ös kanonikus alakból kiindulva, megoldás az MBU-Szimplex algoritmussal:

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{v}_1^*$	$\mathbf{v}_2^*$	$\mathbf{b}$
$\mathbf{u}_1$	1	2	0	1	1	1	0	0	0	110
$\mathbf{v}_1^*$	1	0	2	1	1	0	0	1	0	80
$\mathbf{v}_2^*$	0	2	0	0	1	0	-1	0	1	40
	-1	-2	-3	-2	-3	0	0	0	0	0

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{v}_1^*$	$\mathbf{v}_2^*$	$\mathbf{b}$
$\mathbf{u}_1$	0	2	-2	0	0	1	0	-1	0	30
$\mathbf{x}_1$	1	0	2	1	1	0	0	1	0	80
$\mathbf{v}_2^*$	0	2	0	0	1	0	-1	0	1	40
	0	-2	-1	-1	-2	0	0	1	0	80

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{v}_1^*$	$\mathbf{v}_2^*$	$\mathbf{b}$
$\mathbf{x}_2$	0	1	-1	0	0	1/2	0	-1/2	0	15
$\mathbf{x}_1$	1	0	2	1	1	0	0	1	0	80
$\mathbf{v}_2^*$	0	0	2	0	1	-1	-1	1	1	10
	0	0	-3	-1	-2	1	0	0	0	110

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{v}_1^*$	$\mathbf{v}_2^*$	$\mathbf{b}$
$\mathbf{x}_2$	0	1	-1	0	0	1/2	0	-1/2	0	15
$\mathbf{x}_1$	1	0	2	1	1	0	0	1	0	80
$\mathbf{v}_2^*$	0	0	-2	0	-1	1	1	-1	-1	-10
	0	0	-3	-1	-2	1	0	0	0	110

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{v}_1^*$	$\mathbf{v}_2^*$	$\mathbf{b}$
$\mathbf{x}_2$	1/2	1	0	1/2	1/2	1/2	0	0	0	55
$\mathbf{x}_3$	1/2	0	1	1/2	1/2	0	0	1/2	0	40
$\mathbf{v}_2^*$	1	0	0	1	0	1	1	0	-1	70
	3/2	0	0	1/2	-1/2	1	0	3/2	0	230

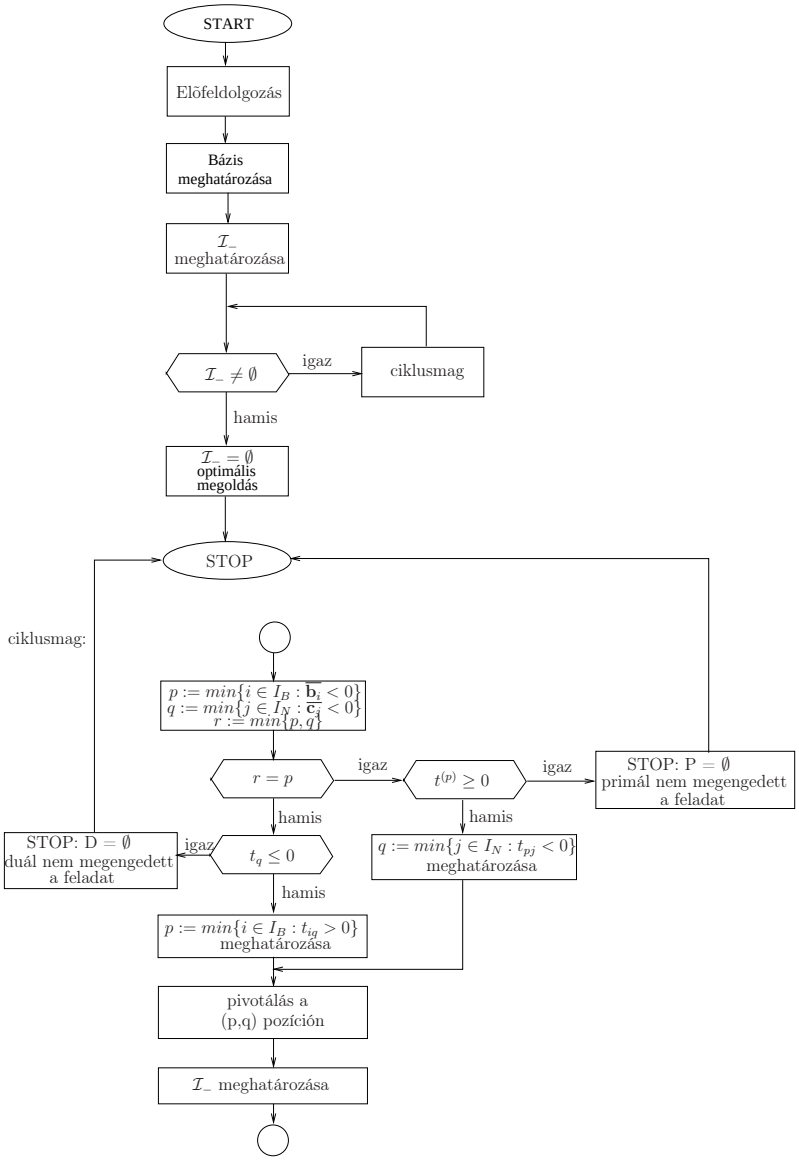
	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{v}_1^*$	$\mathbf{v}_2^*$	$\mathbf{b}$
$\mathbf{x}_2$	0	1	-1	0	0	1/2	0	-1/2	0	15
$\mathbf{x}_5$	1	0	2	1	1	0	0	1	0	80
$\mathbf{v}_2^*$	1	0	0	1	0	1	1	0	-1	70
	2	0	1	1	0	1	0	2	0	270

Az eredeti feladat maximális megoldása az $\mathbf{x}_1 = 0$, $\mathbf{x}_2 = 15$, $\mathbf{x}_3 = \mathbf{x}_4 = 0$, $\mathbf{x}_5 = 80$ és a maximális érték 270.

3.3. Minimál indexes criss-cross módszer

bemenő adatok:
a (P) feladat rövid (bázis) táblája, nem megengedett változók $\mathcal{I}_-$ indexhalmaza
begin
while $(\mathcal{I}_- \neq \emptyset)$ **do**
$p := min\{i \in I_B : \overline{\mathbf{b}_i} < 0\}; q := min\{j \in I_N : \overline{\mathbf{c}_j} < 0\}; r := min\{p, q\};$
if(r = p)
then
if $(t^{(p)} \geq 0)$
then STOP: a P = $\emptyset$;
else legyen $q := min\{j \in I_N : t_{pj} < 0\}$;
endif
else
if$(t_{rq} \leq 0)$
then STOP: a D = $\emptyset$;
else legyen $p := min\{i \in I_B : t_{iq} > 0\}$;
endif
endif

pivotálás a (p,q) pozíción; $I_B := I_B \cup \{q\} \setminus \{p\}$;
határozzuk meg az $\mathcal{I}_-$ halmazt;
endwhile
STOP: optimális megoldásnál vagyunk;
end



3. ábra. Criss-Cross-módszer.

Példa 1:
Oldjuk meg a következő feladatot:

$$\begin{aligned} \min(-\mathbf{x}_1 - 2\mathbf{x}_2 - 4\mathbf{x}_3 - \mathbf{x}_4) \\ \mathbf{x}_1 + \mathbf{x}_3 + \mathbf{x}_4 \leq 5 \\ \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_4 \leq 6 \\ \mathbf{x}_3 \leq 7 \\ \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4 \geq 0 \end{aligned}$$

A (14)-es alakból kiindulva oldjuk meg a fenti feladatot a Criss-Cross algoritmussal:

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{u}_3$	$\mathbf{b}$
$\mathbf{u}_1$	1	0	1	1	1	0	0	5
$\mathbf{u}_2$	1	1	0	1	0	1	0	6
$\mathbf{u}_3$	0	0	1	0	0	0	1	7
	-1	-2	-4	-1	0	0	0	0

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{u}_3$	$\mathbf{b}$
$\mathbf{x}_1$	1	0	1	1	1	0	0	5
$\mathbf{u}_2$	0	1	-1	0	-1	1	0	1
$\mathbf{u}_3$	0	0	1	0	0	0	1	7
	0	-2	-3	0	1	0	0	5

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{u}_3$	$\mathbf{b}$
$\mathbf{x}_1$	1	0	1	1	1	0	0	5
$\mathbf{x}_2$	0	1	-1	0	-1	1	0	1
$\mathbf{u}_3$	0	0	1	0	0	0	1	7
	0	0	-5	0	-1	2	0	7

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{u}_3$	$\mathbf{b}$
$\mathbf{x}_3$	1	0	1	1	1	0	0	5
$\mathbf{x}_2$	1	1	0	1	0	1	0	6
$\mathbf{u}_3$	-1	0	0	-1	-1	0	-1	2
	5	0	0	5	4	2	0	32

A megoldás az $\mathbf{x}_1 = 0$, $\mathbf{x}_2 = 6$, $\mathbf{x}_3 = 5$ és $\mathbf{x}_4 = 0$, és a maximális érték 32. Az algoritmusok során a megoldás nem változhat meg, csak máshogyan, más módszerrel jutunk el oda.

Példa 2:
Oldjuk meg a következő feladatot:

$$\begin{aligned} \min(-\mathbf{x}_1 - 2\mathbf{x}_2 - 3\mathbf{x}_3 - 2\mathbf{x}_4 - 3\mathbf{x}_5) \\ \mathbf{x}_1 + 2\mathbf{x}_2 + \mathbf{x}_4 + \mathbf{x}_5 &\leq 110 \\ \mathbf{x}_1 + 2\mathbf{x}_3 + \mathbf{x}_4 + \mathbf{x}_5 &= 80 \\ 2\mathbf{x}_2 + \mathbf{x}_5 &\geq 40 \\ \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5 &\geq 0 \end{aligned}$$

A fenti (15)-ös kanonikus alakból kiindulva oldjuk meg a feladatot a Criss-Cross algoritmussal:

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{v}_1^*$	$\mathbf{v}_2^*$	$\mathbf{b}$
$\mathbf{u}_1$	1	2	0	1	1	1	0	0	0	110
$\mathbf{v}_1^*$	1	0	2	1	1	0	0	1	0	80
$\mathbf{v}_2^*$	0	2	0	0	1	0	-1	0	1	40
	-1	-2	-3	-2	-3	0	0	0	0	0

Azoknál a feladatoknál, ahol az egyenlőség és a nagyobb-egyenlőség miatt be kellett vezetni *-os, ún. slack-változókat, ott a Criss-Cross algoritmus megkezdése előtt, a Gauss-Jordan eliminációs algoritmussal be kell vinni a *-os változók helyére a bázisba másik elemeket. Ezért kell most az 3. sor 7. elemén csinálni egy báziscserét.

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{v}_1^*$	$\mathbf{v}_2^*$	$\mathbf{b}$
$\mathbf{u}_1$	1	2	0	1	1	1	0	0	0	110
$\mathbf{v}_1^*$	1	0	2	1	1	0	0	1	0	80
$\mathbf{u}_2$	0	-2	0	0	-1	0	1	0	-1	-40
	-1	-2	-3	-2	-3	0	0	0	0	0

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{v}_1^*$	$\mathbf{v}_2^*$	$\mathbf{b}$
$\mathbf{u}_1$	1	2	0	1	1	1	0	0	0	110
$\mathbf{x}_3$	1/2	0	1	1/2	1/2	0	0	1/2	0	40
$\mathbf{u}_2$	0	-2	0	0	-1	0	1	0	-1	-40
	1/2	-2	0	-1/2	-3/2	0	0	3/2	0	120

Miután bevittünk a *-os változók helyére a bázisba két elemet a következő induló táblát kapjuk:

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{b}$
$\mathbf{u}_1$	1	2	0	1	1	1	0	110
$\mathbf{x}_3$	1/2	0	1	1/2	1/2	0	0	40
$\mathbf{u}_2$	0	-2	0	0	-1	0	1	-40
	1/2	-2	0	-1/2	-3/2	0	0	120

Ezt oldjuk meg a Criss-Cross algoritmus szerint.

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{b}$
$\mathbf{x}_2$	1/2	1	0	1/2	1/2	1/2	0	55
$\mathbf{x}_3$	1/2	0	1	1/2	1/2	0	0	40
$\mathbf{u}_2$	1	0	0	1	0	1	1	70
	3/2	0	0	1/2	-1/2	1	0	230

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{b}$
$\mathbf{x}_5$	1	2	0	1	1	1	0	110
$\mathbf{x}_3$	0	-1	1	0	0	-1/2	0	-15
$\mathbf{u}_2$	1	0	0	1	0	1	1	70
	2	1	0	1	0	3/2	0	285

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{b}$
$\mathbf{x}_5$	1	0	2	1	1	0	0	80
$\mathbf{x}_2$	0	1	-1	0	0	1/2	0	15
$\mathbf{u}_2$	1	0	0	1	0	1	1	70
	2	0	1	1	0	1	0	270

A megoldás az $\mathbf{x}_1 = 0$, $\mathbf{x}_2 = 15$, $\mathbf{x}_3 = \mathbf{x}_4 = 0$ és $\mathbf{x}_5 = 80$, és a maximális érték a 270.

Ha a megoldandó feladat olyan alakban adott, hogy csak kisebb-egyenlőséges feltételek szerepelnek benne, akkor a kezdőbázis azokból az újonnan bevezetett változókból áll, amelyeket a kanonikus alak elérése végett vezettünk be. Ellenkező esetben, amikor egyenlőséges és nagyobb-egyenlőséges feltételek is szerepelnek a feladatban, szükség van az ún. slack-változók bevezetésére. Ekkor a Szimplex és az MBU-Szimplex algoritmus esetében a kezdőbázis meghatározása ugyanaz mint az első esetben, míg a Criss-Cross algoritmusnál a slack-változók eltüntetése érdekében a Gauss-Jordan eliminációs algoritmussal be kell vinnünk a bázisba más elemeket, és így kapjuk meg a kezdőbázist.

3.3. Definíció (Szomszédos bázisok). *Az $\mathbb{R}^m$ térnek az $A = [\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n] \in \mathbb{R}^{m \times n}$ mátrix oszlopai által alkotott két bázisát* szomszédos*nak nevezzük, ha mindössze egy vektorban térek el egymástól.*

Az egyik bázist jelölje B és az A mátrixnak azt az oszlopvektorát, amelyben a két bázis eltér $\mathbf{a}$. Ekkor nyilván a B bázis volta miatt

$$\mathbf{a} = B\mathbf{v} \text{ azaz } \sum_{j \in \mathcal{I}_b} v_j \mathbf{a}_j$$

teljesül, valamely $\mathbf{v} \in \mathbb{R}^m$re. Ha a $v_p \neq 0$ akkor az $\mathbf{a}_p$ a következő módon fejezhető ki

$$\mathbf{a}_p = \frac{1}{\mathbf{v}_p}\mathbf{a} - \sum_{j \in \mathcal{I}_B/\{p\}} \frac{\mathbf{v}_j}{\mathbf{v}_p}\mathbf{a}_j.$$

A B_a mátrix, az A mátrix következő oszlopaiból áll: $\{\mathbf{a}_j : j \in \mathcal{I}_B \text{ és } j \neq p\} \cup \{\mathbf{a}\}$. Nyilván, a B_a pontosan akkor nem szinguláris mátrix, ha $\mathbf{v}_p \neq 0$. Ekkor az előző egyenlet $\mathbf{a}_p = B_a\overline{\mathbf{v}}$ alakban írható, ahol
$\overline{\mathbf{v}} = [-\frac{\mathbf{v}_1}{\mathbf{v}_p}, ..., -\frac{\mathbf{v}_{p-1}}{\mathbf{v}_p}, \frac{1}{\mathbf{v}_p}, -\frac{\mathbf{v}_{p+1}}{\mathbf{v}_p}, ..., -\frac{\mathbf{v}_m}{\mathbf{v}_p}]$.

3.4. Definíció (Pivot mátrix). *Definiáljuk a* $\mathcal{P} = [\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_{p-1}, \overline{\mathbf{v}}, \mathbf{e}_{p+1}, ..., \mathbf{e}_m]$ pivot mátrixot. *Ekkor*

$$B = B_a\mathcal{P} \text{ és így } B^{-1} = \mathcal{P}^{-1}B_a^{-1} \text{ azaz } B_a^{-1} = \mathcal{P}B^{-1}$$

3.5. Definíció (LU (Trianguláris) felbontás). *[31] Legyen* $A \in \mathbb{R}^{n \times n}$, *és* $det(A) \neq 0$. *Ekkor A-t fel tudjuk bontani egy alsó- és egy felsőháromszög mátrix szorzatára. Vagyis* $A = L * U$, *ahol*

$$L = \begin{bmatrix} 1 & 0 & \dots & 0 \\ l_{21} & 1 & & \vdots \\ \vdots & & \ddots & 0 \\ l_{n1} & \dots & l_{nn-1} & 1 \end{bmatrix} \qquad U = \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & & \vdots \\ \vdots & & \ddots & u_{n-1n} \\ 0 & \dots & 0 & u_n n \end{bmatrix}$$

L és U elemeit a következőképpen tudjuk kiszámolni:

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj} \qquad i \leq j \qquad (j = 1, \dots, n)$$

$$l_{ij} = \frac{1}{u_{jj}}\left[a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj}\right] \qquad i > j \qquad (i = j+1, \dots, n)$$

3.6. Definíció (QR (Householder) felbontás). *[31] Legyen* $A \in \mathbb{R}^{n \times n}$. *Ekkor az A mátrixot fel tudjuk bontani* $A = Q * R$ *alakra, ahol Q ortogonális mátrix, R pedig felsőháromszög mátrix.*

3.7. Definíció. *[31] Q mátrix* ortogonális*, ha* $QQ^T = I$.
Q tulajdonságai: $Q^T = Q^{-1}, det(Q) = \pm 1$.
Ha Q_1, Q_2 *ortogonális* $\Rightarrow Q_1 * Q_2$ *is ortogonális.*

Tehát egyik bázisról áttérni valamelyik szomszédosra, ha az első bázis az inverz mátrixával adott, akkor egy pivot mátrixszal balról való szorzással térhetünk át a szomszédos bázis inverzére.

3.4. Módosított Szimplex módszer

bemenő adatok:
a kanonikus (P) feladat, azaz a A mátrix a $\mathbf{b}$ és $\mathbf{c}$ vektorok. Adott továbbá egy primál megengedett B bázis, az I_B index halmazzal és a $B = QR$ felbontással Számítsuk ki a $\overline{\mathbf{c}} = \mathbf{x}_B = B^{-1}\mathbf{b} = R^{-1}Q^T\mathbf{b}$ jobboldal vektort és $\mathbf{y}^T = \mathbf{c}_B^T B^{-1} = \mathbf{c}_B R^{-1} Q^T$ duál vektort, a $\overline{\mathbf{c}_N} = \mathbf{c}_N + \mathbf{y}^T A_N$ vektort, valamint a $z = \mathbf{c}_B^T\mathbf{b}$ célfüggvény értéket.
Határozzuk meg az $\mathcal{I}_- := \{i \in \mathcal{I}_N | \overline{\mathbf{c}_i} < 0\}$
begin
while$(\mathcal{I}_- \neq \emptyset)$**do**
legyen $q \in \mathcal{I}_-$ tetszőleges;
if$(t_q = R^{-1}Q^T\mathbf{a}_q \leq 0)$
then STOP: a $D = \emptyset$;
else
$\vartheta := min\{\frac{\overline{\mathbf{b}_j}}{t_{jq}} : j = 1, 2, ..., m$ és $t_{jq} > 0\}$;
$p \in \mathcal{I}_B$ tetszőleges, amelyre $\frac{\overline{\mathbf{b}_k}}{t_{kq}} = \vartheta = \frac{\mathbf{x}_p}{t_{kq}}$ és az $\mathbf{x}_p$ a k. bázisváltozó;
$\mathbf{x}_q = \vartheta$ és $\mathbf{x}_i := \mathbf{x}_i - \vartheta t_{k_i q}$ $i \in \mathcal{I}_B$ és az $\mathbf{x}_i$ a k_i. bázis változó;
$\overline{v_k} := \frac{1}{t_{kq}}$ és $\overline{v_j} := -\overline{v_k} t_{jq}$, j=1,2,...,m $j \neq k$ és QR update;
endif
pivotálás: $\mathcal{I}_B := \mathcal{I}_B \cup \{q\} \setminus \{p\}$;
számítsuk ki az $\mathbf{y}, \mathbf{c}_N$ vektorokat és az $\mathcal{I}_-$ halmazt;
endwhile
optimalitási kritérium: $\mathcal{I}_- = \emptyset$, számítsuk ki az optimális célfüggvény értéket;
end

Megvalósítás:
Az eredeti feladat:

$$A\mathbf{x} = \mathbf{b}$$
$$\mathbf{x} \geq 0$$

alakú.
Tegyük fel, hogy A teljes sorrangú. Ezt, ha nem így van megadva a feladat, akkor le lehet ellenőrizni, pl. a Gauss-Jordan Elimináció segítségével.
A pivot tábla elkészítéséhez szükség van a bázis inverzére. Az invertálás azonban numerikusan instabil, ezért a Módosított Szimplex módszer arra törekszik, hogy elkerülje a bázis invertálását [32].
Általában nincs szükség a teljes pivot táblára, mindig csak egy-egy transzformált sorra, oszlopra, a $\mathbf{c}$ célfüggvényre, és a $\mathbf{b}$ jobboldalra. A pivot tábla módosított oszlopa $\overline{\mathbf{a}_j}$ megoldása a $B\mathbf{x} = \mathbf{a}_j$ egyenletnek, mivel $\mathbf{a}_j = B\overline{\mathbf{a}_i}$. Ezzel a módszerrel el lehet kerülni, hogy mátrixot invertálni kelljen, ha a B bázisnak egy felbontását tároljuk el, mely lehet LU, vagy QR felbontás is. Mi a numerikusan stabilabb QR felbontást fogjuk használni, azonban az implementációk jelentős része az LU felbontást alkalmazza.
Tegyük fel, hogy ismerjük a B bázis QR felbontását. Ekkor a transzformált oszlopokat és sorokat könnyű kiszámolni, mivel $\mathbf{b} = QR\mathbf{x}$. Ennek a megoldása könnyű, mivel $Q^{-1} = Q^T$, R egy felső háromszög mátrix. Így visszahelyettesítéssel megkapható a megoldás. Ennek segítségével a transzformált jobb oldalt és a célfüggvényt is könnyű kiszámolni, de hasonlóan könnyen meghatározhatóak a transzformált sorok és oszlopok is.
Egy báziscsere megfelel annak, hogy a B egyik oszlopát az A egy másik oszlopára cseréljük. Az új bázis QR felbontását hatékonyan meghatározhatjuk az új oszlop és az előző oszlop QR felbontásából. Ezt a lépést a QR felbontás update-elésének nevezzük.
Természetesen az update-elések között is felgyülemelhet a numerikus kerekítési hiba. Emiatt a QR felbontás minőségét érdemes rendszeres időközönként ellenőrizni, vagyis megvizsgálni, hogy a szorzatuk mennyire pontosan adja vissza a B bázist. A mennyiben az eltérés már egy adott küszöbszint felett van (pl. valamilyen normában), akkor a B bázisnak elölről elkészítjük a QR felbontását. Ezt a lépést hívjuk újrainvertálásnak.
Az algoritmusban a báziscserék, megállási kritériumok természetesen ugyanazok, mint a Primál Szimplex módszernél.

3.5. MPS fájlformátum

Az MPS (Mathematical Programming System) formátumot eredetileg az IMB vezette be a lineáris és egészértékű programozási modellek leírására. Ezekben az időkben még lyukkártyát használtak, ezért alakult ki egy fix oszlopszerkezet, amelyben minden információt a megfelelő oszlopba kellett írni. Eredetileg az ún. Fix MPS formátumot használták, de azóta bevezettek pár módosítást, és így alakult ki az ún. Free MPS formátum. A legfontosabb dolog amit érdemes tudni róla, hogy ez egy oszloporientált formátum, ellen-

tétben a modell tényleges felírásával, és itt minden változó, feltétel ... stb. kap egy nevet.

A mezők a 2., 5., 15., 25., 40., és 50. oszlopban kezdődnek. Az egyes szakaszok ún. header card-al kezdődnek, melyeket sajátossága, hogy az első oszlopban kezdőnek. Általában ezek a fájlok csak nagybetűket tartalmaznak, de header card-ok (címkék) kivételével bárhol lehet kisbetűket is használni. A változóknak, és a korlátozó feltételeknek adott nevek teljesen lényegtelenek a megoldás szempontjából. Javasolt könnyen értelmezhető neveket adni.

Egy MPS fájlban, semmi nem utal az optimalizálás irányára, és nincs alapértelmezett irány sem. Bizonyos lineáris programozási rendszerek maximalizálnak ha nem adunk meg egyebet, mások pedig minimalizálnak, és megint mások pedig először elmentik az adatokat és ha nincs megadva alapértelmezett irány, akkor megkövetelik, hogy például hívási paraméterként adjuk meg az optimalizálás irányát. Ha a mi modellünk minimalizál, de a program maximalizálásra van beállítva - vagy fordítva - akkor a célfüggvényünket szorozzuk be -1-el. Ekkor a kapott optimális célfüggvény érték -1-szerese lesz annak amit mi keresünk, tehát ismét meg kell szorozni -1-el, de a változók értékei megfelelőek, vagyis azokat nem kell megszorozni.

Amelyik sor elejére *-ot teszünk, azt kommentnek tekinti a feldolgozó program. Az általunk adott neveknél érdemes arra odafigyelni, hogy szóközt ne tartalmazzanak, mert nincs automatikus névigazítás, tehát például a SOR1 és a SOR1 nem ugyanazt a változót jelenti. (Az utóbbi végén két szóköz van.) Néhány optimalizáló nem is engedélyezi a szóközt a nevekben.

Az MPS formátum szakaszokból (mezőkből) áll, melyek sorrendje kötött. A következőkben részletezzük az egyes mezőket:
A `NAME` címke után megadhatjuk a feladat nevét.
A `ROWS` mezőben definiáljuk a feltételek neveit, (pl. célfüggvény, feltétel1,) ahogy azt majd később két példában is bemutatjuk és hogy milyen relációt tartalmaznak. Az `E` jelöli az egyenlőséget, az `L` a kisebb-egyenlőséget ($\leq$), a `G` a nagyobb-egyenlőséget ($\geq$), és végül az `N` ha nincs megkötés az adott feltételre. Vagyis `N`-el jelöljük a célfüggvény sorát is. Több célfüggvény esetén mindegyiket `N`-el jelöljük, és máshol adjuk meg hogy melyiket használjuk, azonban ez már a Free MPS formátumhoz tartozik amit később fogunk ismertetni.
A fájl legnagyobb része a `COLUMNS` rész. Ezen a részen írjuk le az A mátrixot. Itt a bejegyzéseket szigorúan egymás után kell írni, de a sorrend itt is lényegtelen. Két lehetőségünk van az adatok bevitelére. Vagy sorfolytonosan

írjuk a megfelelő oszlopokba az adatokat, vagy minden sorba csak egy változóra vonatkozó adatokat írunk. Az áttekinthetőség végett a második ajánlott. Csak a nem nulla elemeket kell ide beírni. Mivel a tapasztalatok azt mutatják, hogy úgy numerikusan hatékonyabb, ezért a kereskedelmi implementációk nem hozzák a feladatokat kanonikus alakra, hanem az eredeti alakban próbálják meg megoldani őket [22]. (Vagyis ezért a megoldó algoritmusokat próbálják úgy általánosítani, hogy tetszőleges feladatot az eredeti formában tudjanak megoldani.) Természetesen lehet olyan feladat is, amelyben vegyesen szerepelnek egész és nem feltétlenül egész változók is, de ilyen a mi feladatainknál nem fog előfordulni.
Az `RHS` részben definiáljuk a jobboldal vektorokat. Ebből is lehet több.
A `BOUNDS` rész megadása nem kötelező. Itt adjuk meg a változóinkra vonatkozó korlátozó feltételeket, ahelyett hogy a mátrixban definiálnánk extra sorokat. Az `UP` azt jelenti, hogy a változónál nagyobb, a `LO` pedig, hogy kisebb az az érték, mint amit a változó neve utáni oszlopba adunk meg. Az `LI` például azt jelenti mint az `LO`, csak egész értékekre vonatkozik. Természetesen vannak még egyéb beállítási lehetőségek is ebben a részben. Ha itt nem adunk meg semmit, akkor az alapértelmezés az $\mathbf{x} \geq 0$. Ezért mi sem fogunk ide semmit írni a példáinkban.
A másik opcionális címke a `RANGES`. Itt adhatunk meg olyan korlátozásokat, amelyek az adott feltételhez tartozó jobboldal mozgási szabadságát jellemzi.
És végül a lezáró címke az `ENDATA`.

A Free MPS formátum nagyon hasonlít a Fixed MPS formátumhoz, csak néhány korlátozástól eltekint és néhány új kényelmi szolgáltatást vezet be. A mezőknek nincsen kötött helyük. Csak annyi a megkötés, hogy az első oszlopot kivéve minden mezőt el kell választani az utána következőtől egy vagy több szóközzel, de az oszlopok sorrendjének meg kell egyeznie a Fix MPS-nél leírtakkal. Legfontosabb változtatás a Fix MPS-hez képest, hogy itt nem tartalmazhatnak szóközt a változók nevei. Tulajdonképpen nincs egy általános formája ennek a formátumnak, mert minden implementer saját formát használ. Némelyik engedélyezi a szóközöket a nevekben, némelyik megköveteli, hogy a név legalább 8 karakterből álljon, míg van olyan is, amelyik csak az első 6 karaktert olvassa be, a többit figyelmen kívül hagyja.
Némely megoldó program további parancsszavakat vesz hozzá a Free formátumhoz, amely például meghatározza az optimalizálás irányát. Ez az új opcionális címke a `OBJSENSE`, amit a `NAME` és a `ROWS` címkék között kell megadni. Az ezt követő sorban egy szóközt kihagyva adhatjuk meg, ha maximalizálni szeretnénk. A lehetséges értékek: `MAXIMIZE, MAX, MINIMIZE, MIN`. Ha nem adjuk ezt meg, akkor például az lp_solve program minimalizál. A legtöbb programnak szintén a minimalizálás az alapértelmezett.

Más programok egy másik "szabványt" vezettek be arra az esetre, ha több célfüggvénnyel szeretnénk dolgozni. Ekkor az `OBJNAME` címke után adhatjuk meg a használni kívánd célfüggvény nevét. Ha nem adtunk meg itt semmit, akkor a program a `ROWS` részben, az első `N`-el jelölt sort fogja célfüggvényként használni. Ezt is a `NAME` és a `ROWS` címkék között kell megadni.

A feljebb ismertetett két feladat:

$$\begin{aligned} \max(\mathbf{x}_1 + 2\mathbf{x}_2 + 4\mathbf{x}_3 + \mathbf{x}_4) \\ \mathbf{x}_1 + \mathbf{x}_3 + \mathbf{x}_4 \leq 5 \\ \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_4 \leq 6 \\ \mathbf{x}_3 \leq 7 \\ \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4 \geq 0 \end{aligned}$$

$$\begin{aligned} \max(\mathbf{x}_1 + 2\mathbf{x}_2 + 3\mathbf{x}_3 + 2\mathbf{x}_4 + 3\mathbf{x}_5) \\ \mathbf{x}_1 + 2\mathbf{x}_2 + \mathbf{x}_4 + \mathbf{x}_5 \leq 110 \\ \mathbf{x}_1 + 2\mathbf{x}_3 + \mathbf{x}_4 + \mathbf{x}_5 = 80 \\ 2\mathbf{x}_2 + \mathbf{x}_5 \geq 40 \\ \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5 \geq 0 \end{aligned}$$

a következőképpen néz ki a már ismertetett MPS formátumban:

```
NAME PELDA1
OBJSENSE
 MAX
ROWS
 N  CELFGV
 L  FELT1
 L  FELT2
 L  FELT3
COLUMNS
    XEGY      CELFGV   1
    XEGY      FELT1    1
    XEGY      FELT2    1
    XKETTO    CELFGV   2
    XKETTO    FELT2    1
    XHAROM    CELFGV   4
    XHAROM    FELT1    1
```

```
    XHAROM    FELT3    1
    XNEGY     CELFGV   1
    XNEGY     FELT1    1
    XNEGY     FELT2    1
RHS
    RHS1      FELT1    5
    RHS1      FELT2    6
    RHS1      FELT3    7
ENDATA

NAME PELDA2
OBJSENSE
 MAX
ROWS
 N  CELFGV
 L  FELT1
 E  FELT2
 G  FELT3
COLUMNS
    XEGY      CELFGV     1
    XEGY      FELT1      1
    XEGY      FELT2      1
    XKETTO    CELFGV     2
    XKETTO    FELT1      2
    XKETTO    FELT3      2
    XHAROM    CELFGV     3
    XHAROM    FELT2      2
    XNEGY     CELFGV     2
    XNEGY     FELT1      1
    XNEGY     FELT2      1
    XOT       CELFGV     3
    XOT       FELT1      1
    XOT       FELT2      1
    XOT       FELT3      1
RHS
    RHS1      FELT1    110
    RHS1      FELT2     80
    RHS1      FELT3     40
ENDATA
```

A fentiek csak egy rövid áttekintését adják az MPS formátumnak, részletesebb információt a [23]-as, [26]-os könyvekben találunk.

4. Implementáció

A következőkben bemutatjuk az általunk készített Matlab implementációk legfontosabb tulajdonságait, ismertetjük az egyes függvények leírását, illetve hogy a működés során mely eljárás mely eljárásokat illetve függvényeket hív meg.

4.1. Függvényleírások

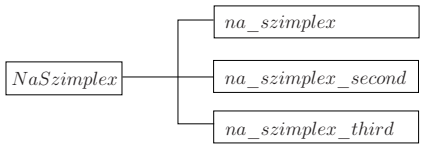
Az alábbiakban felsoroljuk az általunk használt függvényket és pár mondatban megadjuk a működésük leírását is.

`NaSzimplex`	Kiszámítja az A mátrix kondíciószámát, rangját, megad egy kezdőbázist, majd meghívja a következő három eljárást.
`na_szimplex`	Megvalósítja a Szimplex algoritmus első fázisát.
`na_szimplex_second`	Ha valaki bennmaradt a kezdőbázisból, itt visszük ki.
`na_szimplex_third`	Megvalósítja a Szimplex algoritmus második fázisát.
`NaMBU`	Kiszámítja az A mátrix kondíciószámát, rangját, megad egy kezdőbázist, majd meghívja a következő eljárásokat.
`na_MBU_szimplex`	Megvalósítja az MBU-Szimplex algoritmus első fázisát.
`na_MBU_third`	Megvalósítja az MBU-Szimplex algoritmus második fázisát.
`NaCC`	Megvalósítja a Criss-Cross algoritmust.
`CreateBaseMatrix()`	Megadja az A mátrixban a Bázis mátrixot.
`GetPrices()`	Kiszámolja az éppen aktuális célfüggvényt a QR felbontás és a Bázis alapján
`GetRHS()`	Kiszámolja a jobboldalt a QR felbontás és az eddigi b alapján.
`GetSol()`	Kiszámolja a pillanatnyi megoldást az QR felbontás, a Bázis, az A mátrix és a jobboldal alapján.
`GetTransformedColumn()`	Visszaadja a transzformált oszlopot az A mátrix QR felbontása alapján.
`GetTransformedRow()`	Visszaadja a transzformált sort az A mátrix QR felbontása alapján.

4.2. Összefüggések

Az következő ábrák diagrammok bemutatják, hogy az egyes eljárások milyen más eljárásokat használnak a futásuk során.

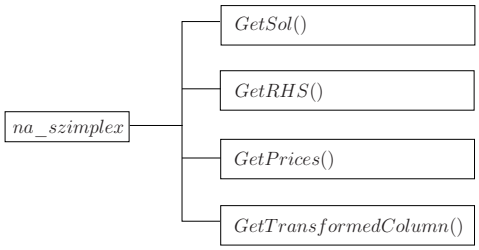
Ahogy azt a lenti ábra is mutatja, a `NaSzimplex` eljárást elindítva a `na_szimplex`, a `na_szimplex_second` és a `na_szimplex_third` eljárásokat használja.



4. ábra. NaSzimplex eljárás.

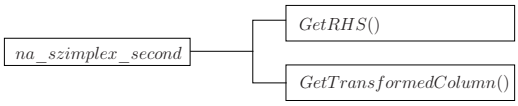
A következőkben azt részletezzük, hogy az egyes eljárások milyen függvényeket használnak a működésük során. Látható lesz, hogy összesen hat féle függvényünk van, de nem mindegyik eljárás használja mindet, illetve van amit csak az egyik használ, míg van amelyiket mindegyik eljárás meghívja a futása során.

A `na_szimplex` eljárás a `GetRHS()`, a `GetPrices()`, a `GetTransformedColumn()` és a `GetSol()` függvényeket használja.



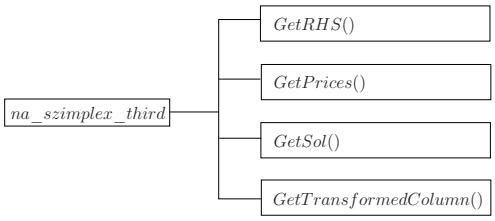
5. ábra. na_szimplex eljárás.

A `na_szimplex_second` eljárás a `GetRHS()` és a `GetTransformedColumn()` függvényeket használja. Mivel ezt az eljárást használja a `NaMBU` eljárás is, ezért ennek kifejtését ott nem ismételjük meg.



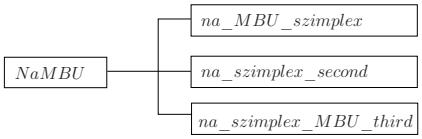
6. ábra. na_szimplex_second eljárás.

A `na_szimplex_third` eljárás a `GetRHS()`, a `GetPrices()`, a `GetTransformed-Column()` és a `GetSol()` függvényeket használja.



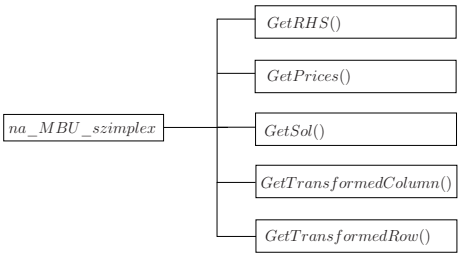
7. ábra. na_szimplex_third eljárás.

A `NaMBU` eljárás a `na_MBU_szimplex`, a `na_szimplex_second`, és a `na_szim-plex_MBU_third` eljárásokat használja a futása során.



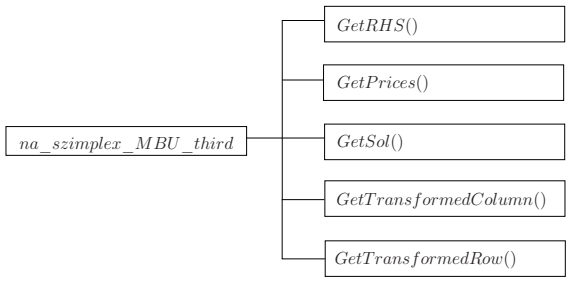
8. ábra. NaMBU eljárás.

A fentieket részletezve, a `na_MBU_szimplex` eljárás az összes függvény használja a `CreateBaseMatrix()` kivételével.



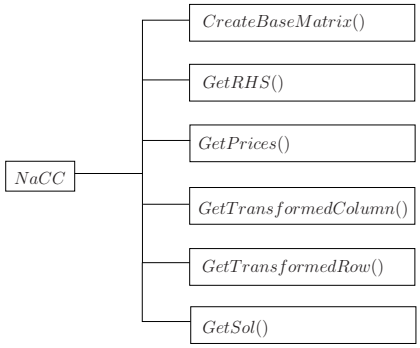
9. ábra. na_MBU_szimplex eljárás.

A `na_MBU_third` eljárás is használja a következő függvényeket: `GetRHS()`, `GetPrices()`, `GetTransformedColumn()`, `GetTransformedRow()`, `GetSol()`.



10. ábra. na_MBU_third eljárás.

A fentiekkel ellentétben a `NaCC` eljárás nem hív meg másik eljárást, hiszen míg azok több fázisból álltak, addig a a Criss-Cross algoritmusnál nincs értelme fázisokról beszélni. Ezért ez csak a `CreateBaseMatrix()`, a `GetRHS()`, a `GetPrices()`, a `GetTransformedColumn()` és a `GetTransformedRow()` függvényekkel van kapcsolatban.



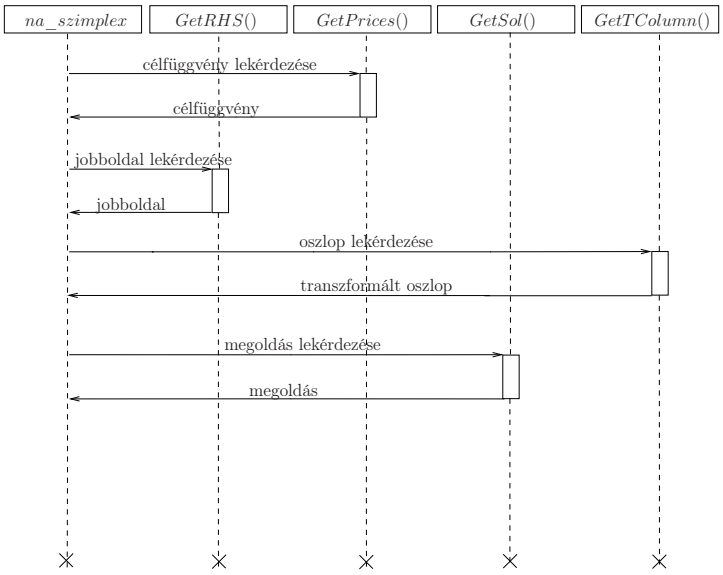
11. ábra. NaCC eljárász.

4.3. Futási esetek

A következő szekvencia diagrammok azt mutatják be, hogy az egyes eljárások, a futásuk során milyen sorrendben hívják meg az egyes függvényeket. Természetesen a következő diagrammok csak a futás egy részét ábrázolják.
Ezeken a diagramokon, a `GetTransformedColumn()`, `GetTColumn()`, és a `GetTransformedRow()`, `GetTRow()` néven szerepel.

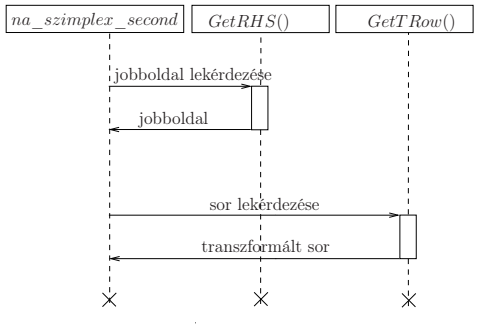
Ahogy azt már a fenti diagramon láttuk, a `NaSzimplex` futtatásakor, meghívódik a `na_szimplex`, a `na_szimplex_second`, és a `na_szimplex_third` eljárás. Most ezeket az eljárásokat fogjuk részletesebben bemutatni.

A `na_szimplex` eljárás a futása során a következő függvényeket használja: `GetRHS()`, `GetPrices()`, `GetSol()`, `GetTColumn()`. Az eljárás az indulása után, egyéb műveletek elvégzése után először lekéri a célfüggvény értékét, majd miután ezt megkapta a `GetPrices()` függvénytől, lekérdezi a **b** vektor (jobboldal) értékét a `GetRHS()`-től. Ezekután a `GetTColunm()` függvény segítségével megkapja a keresett oszlopot, és végül a `GetSol()` függvény megadja a pillanatnyi megoldást.
Mivel ez csak az eljárás futásának egy részét tükrözi, ezért a fentiek után ismét lekérheti a `GetPrices()` függvénnyel a célfüggvény értékét.



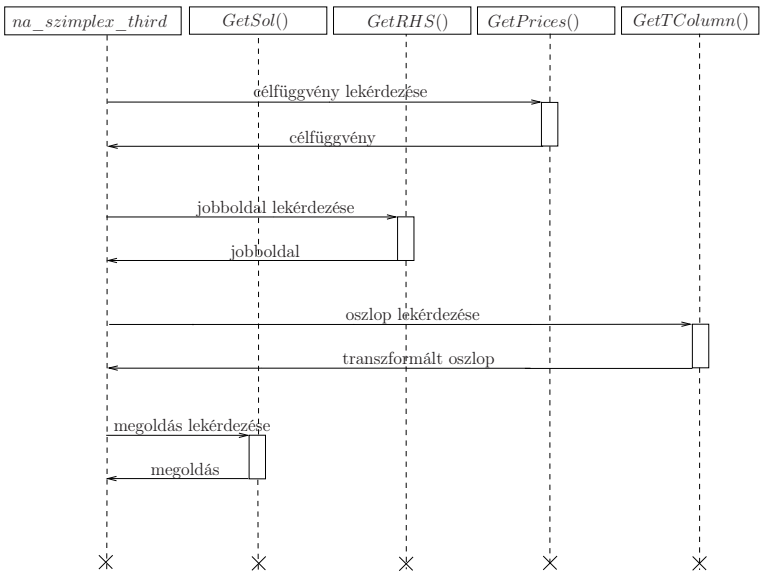
12. ábra. na_szimplex szekvencia diagramja.

A `na_szimplex_second` eljárás a futása során a következő függvényeket használja: `GetRHS()` és `GetTRow()`. Először megkapja a `GetRHS()` függvénytől a jobboldal pillanatnyi értékét, majd a `GetTRow()` függvény megadja a transzformált sort.



13. ábra. na_szimplex_second szekvencia diagramja.

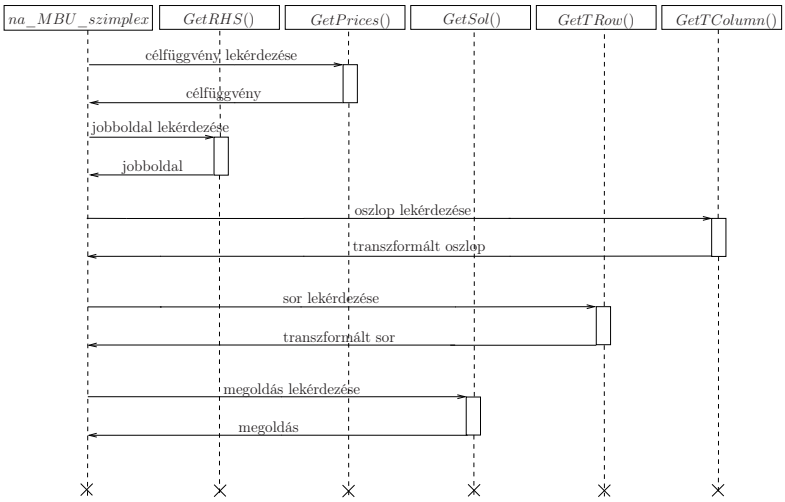
A `na_szimplex_third` eljárás a futása során a következő függvényeket használja: `GetSol()`, `GetRHS()`, `GetPrices()`, `GetTColumn()`. Kezdetben lekéri a `GetPrices()` függvénytől a célfüggvény pillanatnyi értékét, utána `GetRHS()`-től a jobboldalt, majd a `GetTColumn()` megadja a transzformált oszlopot, és végül `GetSol()` pedig a pillanatnyi megoldást.



14. ábra. na_szimplex_third szekvencia diagramja.

Láttuk, hogy a `NaMBU` futásakot meghívódik a `na_MBU_szimplex`, a `na_szimplex_second`, és a `na_MBU_third` eljárás. Most ezek futását tekinthetjük meg az alábbi szekvencia diagramokon.

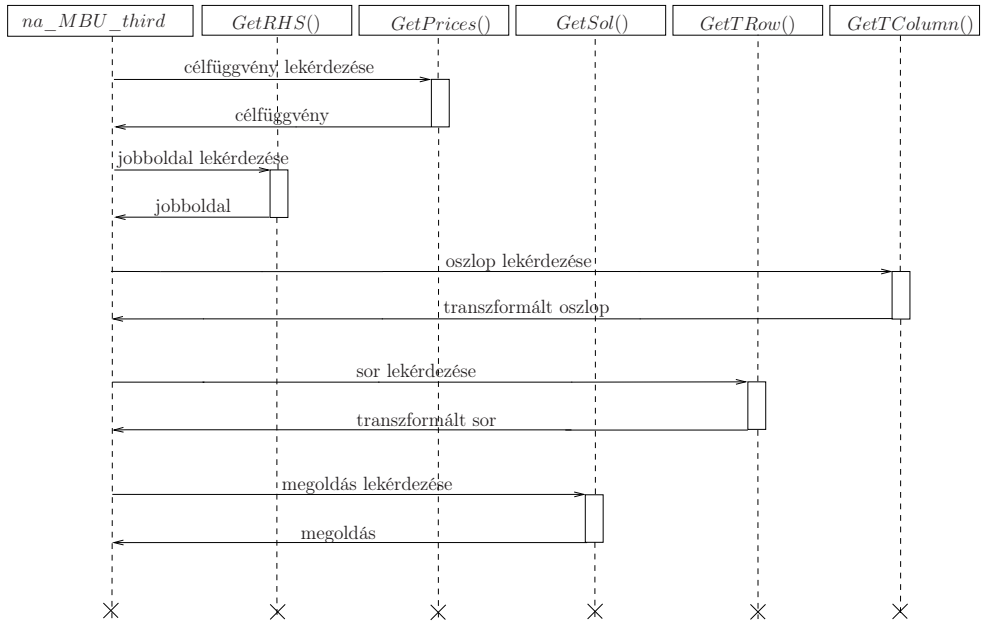
A `na_MBU_szimplex` eljárás a futása során a következő függvényeket használja: `GetRHS()`, `GetPrices()`, `GetSol()`, `GetTRow()`, és `GetTColumn()`. Először megkapjuk a `GetPrices()` függvénytől a célfüggvény értékét, utána a `GetRHS()` függvény megadja a jobboldalt. Egyéb műveletek után a `GetTColumn()` függvénytől lekérdezi a transzformált oszlopot, a `GetTRow()` függvénytől a transzformált sort, és végül a `GetSol()` függvény megadja a pillanatnyi megoldás értékét.



15. ábra. na_MBU_szimplex szekvencia diagramja.

Mivel a `NaMBU` eljárás is használja a `na_szimplex_second` eljárást, ezért azt itt nem ismételjük meg mégegyszer.

A `na_MBU_Third` eljárás a futása során a következő függvényeket használja: `GetRHS()`, `GetPrices()`, `GetSol()`, `GetTRow()`, és `GetTColumn()`. Ez az eljárás, a `na_MBU_szimplex`-hez hasonlóan először lekéri a célfüggvényt a `GetPrices()` függvénytől, utána a jobboldalt a `GetRHS()` függvénytől, majd a `GetTColumn()` segítségével megkapja az oszlopot, a `GetTRow()` függvény megadja a keresett sort, és végül a `GetSol()` függvény visszaadja a pillanatnyi megoldást.

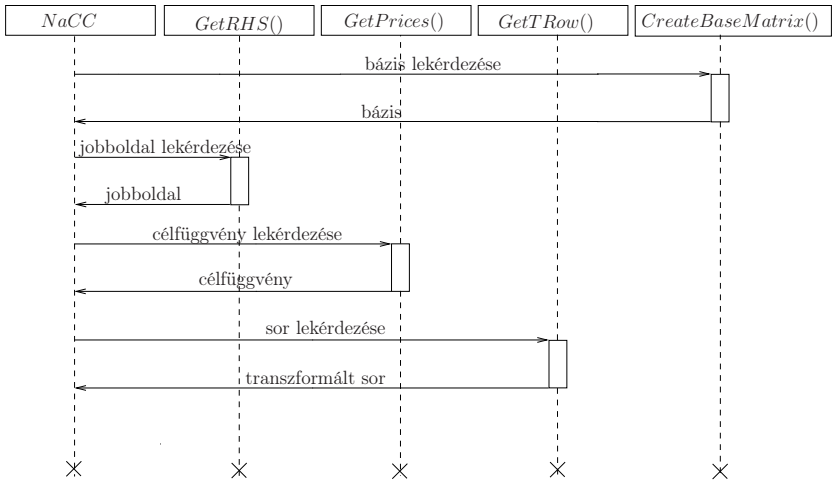


16. ábra. na_MBU_third szekvencia diagramja.

Végül bemutatjuk, hogy a `NaCC` eljárás hogyan hívhat meg függvényeket a futása során. Itt a függvény törzsében levő elágazástól függően két féle függvényhívás lehet, ezért most két diagrammal írjuk le a lehetséges futásokat.
Ahogy azt már a Criss-Cross algoritmus leírásnál is láttuk, az r változó kétféle értéket vehet fel, attól függően, hogy a p, vagy a q változó értékét veszi fel.

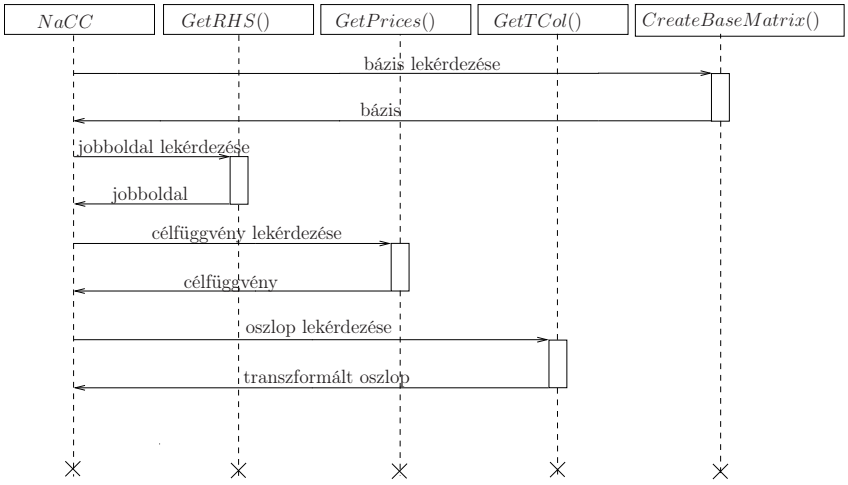
A futása során így a `NaCC` eljárás először lekérdezi a bázist a `CreateBaseMatrix()` segítségével, majd a jobboldalt a `GetRHS()` függvény adja meg, és a pillanatnyi célfüggvényt a `GetPrices()` függvény.

Ha a futás során az r változó a p értékét veszi fel, vagyis ha a p sorában az első negatív helyen akarunk pivotálni, akkor a `GetTRow()` függvénnyel lekérdezzük a p-nek a sorát.



17. ábra. NaCC_szekvencia diagramja.

Különben pedig ha az r változó a q értékét veszi fel, vagyis ha a q sorában az első pozitív helyen akarunk pivotálni, akkor a `GetTColumn()` függvénnyel lekérdezzük a q-nak az oszlopát.



18. ábra. NaCC_szekvencia diagramja.

5. Tesztfeladatok

Az alábbiakban ismertetjük azokat a tesztfeladatokat, amelyeket a NETLIB adatbázisból nyertünk. Ezek közismert lineáris programozási tesztfeladatok, amelyek között mindenféle méretű és típusú megtalálható. Ezt az adatbázist kifejezetten olyan célból hozták létre, hogy ismert és kevésbé ismert példákat gyűjtsenek össze általános tesztelési célokra.
Természetesen itt nem csak lineáris programozási feladatok találhatóak, hanem többféle konvertáló szoftver és más feladatokhoz való tesztadatok is fellelhetőek. A NETLIB a matematikai szoftverek, tanulmányok és adatbázisok gyűjteménye.

5.1. Eredeti tesztfeladatok

A tesztfeladatokat az [1]-es adatbázisból nyertük. A következő táblázatban összefoglaljuk a feladatok adatait. Megadjuk minden feladathoz hogy hány sora, oszlopa, nemnulla eleme van, és mi az optimuma, majd hogy mindezt az CPLEX és a Matlab Linprog nevű függvénye hány iteráció alatt érte el.
Ahogy az látszik is, az egyes feladatok nagyon nagy méretűek, ezért a vannak olyan feladatok amelyeket a Linprog függvénnyel, nem tudtunk megoldani, a méretük miatt.
Azért adtuk itt meg a [1]-ben elérhető összes tesztfeladatot, hogy érzékeltessük, hogy az általunk készített implementációk mekkora részét képesek megoldani. Ahogy azt az ezt követő táblázat is mutatja, az itt következő feladatoknak csupáncsak a harmadát tudtuk megoldani a saját implementációinkkal.

	Eredeti feladat				**Iteráció szám**	
Név	**Sor**	**Oszlop**	**Nem-nulla**	**Optimum**	**CPLEX**	**Lin-prog**
25FV47	822	1571	11127	5,502E+03	2302 (785)	26
80BAU3B	2263	9799	29063	9,872E+05	8808 (1689)	
ADLITTLE	57	97	465	2,255E+05	94 (16)	11
AFIRO	28	32	88	-4,648E+02	7 (0)	8
AGG	489	163	2541	-3,599E+07	76 (43)	20
AGG2	517	302	4515	-2,024E+07	136 (35)	18
AGG3	517	302	4531	1,031E+07	138 (23)	17
BANDM	306	472	2659	-1,586E+02	216 (87)	17
BEACONFD	174	262	3476	3,359E+04	1 (0)	12
BLEND	75	83	521	-3,081E+01	81 (0)	12
BNL1	644	1175	6129	1,978E+03	2169 (1899)	28
BNL2	2325	3489	16124	1,811E+03	4532 (2880)	
BOEING1	351	384	3865	-3,352E+02	550 (172)	22
BOEING2	167	143	1339	-3,150E+02	156 (93)	19
BORE3D	234	315	1525	1,373E+03	22 (10)	21

	Eredeti feladat				Iteráció szám	
Név	**Sor**	**Oszlop**	**Nem-nulla**	**Optimum**	**CPLEX**	**Lin-prog**
BRANDY	221	249	2150	1,519E+03	183 (90)	18
CAPRI	272	353	1786	2,690E+03	281 (203)	18
CYCLE	1904	2857	21322	-5,226E+00	874 (0)	
CZPROB	930	3523	14173	2,185E+06	859 (311)	
D2Q06C	2172	5167	35674	1,228E+05	9822 (1330)	
D6CUBE	416	6184	43888	3,155E+02	26932 (2414)	
DEGEN2	445	534	4449	-1,435E+03	2100 (692)	18
DEGEN3	1504	1818	26230	-9,873E+02	8101 (6508)	86
DFL001	6072	12230	41873	1,127E+07	63963 (13039)	
E226	224	282	2767	-1,875E+01	304 (39)	20
ETAMACRO	401	688	2489	-7,557E+02	712 (406)	25
FFFFF800	525	854	6235	5,557E+05	424 (182)	86
FINNIS	498	614	2714	1,728E+05	387 (155)	23
FIT1D	25	1026	14430	-9,146E+03	1074 (0)	17
FIT1P	628	1677	10894	9,146E+03	959 (627)	52
FIT2D	26	10500	138018	-6,846E+04	14289 (0)	
FIT2P	3001	13525	60784	6,846E+04	17530 (2425)	
FORPLAN	162	421	4916	-6,642E+02		
GANGES	1310	1681	7021	-1,096E+05	433 (35)	20
GFRD-PNC	617	1092	3467	6,902E+06	509 (284)	18
GREENBEA	2393	5405	31499	-7,246E+07	6140 (1810)	
GREENBEB	2393	5405	31499	-4,302E+06	5419 (1718)	
GROW15	301	645	5665	-1,069E+08	1125 (0)	23
GROW22	441	946	8318	-1,608E+08	1436 (0)	23
GROW7	141	301	2633	-4,779E+07	270 (0)	20
ISRAEL	175	142	2358	-8,966E+05	190 (1)	21
KB2	44	41	291	-1,750E+03	49 (0)	16
LOTFI	154	308	1086	-2,526E+01	0 (0)	23
MAROS	847	1443	10006	-5,806E+04	1005 (497)	25
MAROS-R7	3137	9408	151120	1,497E+06	3566 (754)	
MODSZK1	688	1620	4158	3,206E+02	639 (222)	25
NESM	663	2923	13988	1,408E+07	3599 (914)	
PEROLD	626	1376	6026	-9,381E+03	2786 (1742)	74
PILOT	1442	3652	43220	-5,574E+02	7383 (3737)	
PILOT.JA	941	1988	14706	-6,113E+03	6346 (2718)	86
PILOT.WE	723	2789	9218	-2,720E+06	2665 (472)	86
PILOT4	411	1000	5145	-2,581E+03	907 (282)	
PILOT87	2031	4883	73804	3,017E+02	9862 (4423)	
PILOTNOV	976	2172	13129	-4,497E+03	3370 (1879)	29

	Eredeti feladat				Iteráció szám	
Név	**Sor**	**Oszlop**	**Nem-nulla**	**Optimum**	**CPLEX**	**Lin-prog**
QAP8	913	1632	8304	2,035E+02	6175 (1629)	3
QAP12	3193	8856	44244	5,229E+02	58085 (6249)	
QAP15	6331	22275	110700	1,041E+03		
RECIPE	92	180	752	-2,666E+02	21 (6)	11
SC105	106	103	281	-5,220E+01	65 (0)	10
SC205	206	203	552	-5,220E+01	156 (0)	11
SC50A	51	48	131	-6,458E+01	27 (0)	10
SC50B	51	48	119	-7,000E+01	33 (0)	8
SCAGR25	472	500	2029	-1,475E+07	320 (208)	15
SCAGR7	130	140	553	-2,331E+06	75 (43)	12
SCFXM1	331	457	2612	1,842E+04	269 (125)	18
SCFXM2	661	914	5229	3,666E+04	569 (291)	19
SCFXM3	991	1371	7846	5,490E+04	953 (491)	20
SCORPION	389	358	1708	1,878E+03	58 (34)	17
SCRS8	491	1169	4029	9,043E+02	502 (50)	23
SCSD1	78	760	3148	8,667E+00	208 (76)	11
SCSD6	148	1350	5666	5,050E+01	476 (190)	
SCSD8	398	2750	11334	9,050E+02	1777 (791)	12
SCTAP1	301	480	2052	1,412E+03	195 (97)	17
SCTAP2	1091	1880	8124	1,725E+03	542 (231)	20
SCTAP3	1481	2480	10734	1,424E+03	746 (335)	
SEBA	516	1028	4874	1,571E+04	6 (2)	32
SHARE1B	118	225	1182	-7,659E+04	137 (73)	23
SHARE2B	97	79	730	-4,157E+02	97 (61)	12
SHELL	537	1775	4900	1,209E+09	371 (158)	14
SHIP04L	403	2118	8450	1,793E+06	235 (73)	12
SHIP04S	403	1458	5810	1,799E+06	157 (38)	13
SHIP08L	779	4283	17085	1,909E+06	445 (131)	
SHIP08S	779	2387	9501	1,920E+06	273 (77)	14
SHIP12L	1152	5427	21597	1,470E+06	730 (279)	
SHIP12S	1152	2763	10941	1,489E+06	356 (123)	
SIERRA	1228	2036	9252	1,539E+07	406 (147)	
STAIR	357	467	3857	-2,513E+02	336 (205)	28
STANDATA	360	1075	3038	1,258E+03	36 (8)	86
STANDMPS	468	1075	3686	1,406E+03	205 (93)	20
STOCFOR1	118	111	474	-4,113E+04	31 (8)	17
STOCFOR2	2158	2031	9492	-3,902E+04	1052 (405)	
STOCFOR3	16676	15695	74004	-3,998E+04	9833 (3017)	
TRUSS	1001	8806	36642	4,588E+05	13094 (2294)	

	Eredeti feladat				Iteráció szám	
Név	**Sor**	**Oszlop**	**Nem-nulla**	**Optimum**	**CPLEX**	**Lin-prog**
TUFF	334	587	4523	2,921E-01	356 (205)	86
VTP.BASE	199	203	914	1,298E+05	27 (23)	17
WOOD1P	245	2594	70216	1,443E+00	466 (245)	20
WOODW	1099	8405	37478	1,304E+00	1285 (367)	

5.2. Általunk készített programok

A következőkben bemutatjuk, hogy az általunk készített Matlab programmal milyen eredményeket kaptunk a tesztpéldák futtatása során. Minden egyes példánál megadjuk az iterációszámot, és a kapott értéket. Egyes cellák azért maradtak üresen, mert azokat az adott algoritmus nem tudta kiszámolni. Ez legtöbbször a Criss-Cross algoritmus esetén fordult elő. A Szimplex és az MBU algoritmus esetén az első és a második fázis iterációszámai szerepelnek az adott cellában.
Ebben az esetben az adatok nagyság szerint növekvően vannak rendezve, míg az előző táblázatban névsor szerint voltak a tesztadatok felsorolva. Ahol csak egy iteráció szerepel, ott az adott program csak addig jutott el és ott valamilyen hibával leállt.

A Criss-Cross algoritmusnál jelentősen adódtak olyan esetek, amikor a Matlab nem tekintette a mátrixunkat regulárisnak. Ezeket az eseteket nem tüntettük fel a táblázatban.

Jelen táblázatban a CC-vel jelöltük a Criss-Cross algoritmust. Ennek az oszlopában a NaN a Not a Number jelölése, és ezt olyan esetekben kaptuk, amikor a kerekítési hibák miatt olyan kis számok jöttek ki, hogy azt a Matlab a hányadostesztnél már 0/0 osztásnak vette.

	Iterációszám			Érték		
Név	**Szimplex**	**MBU**	**CC**	**Szimplex**	**MBU**	**CC**
AFIRO	28+1	37+3		-4,648E+02	-4,648E+02	
KB2	108+48	104+33		-1,750E+03	-1,750E+03	
SC50A	54+5	70+5	5	-6,458E+01	-6,458E+01	-6,458E+01
SC50B	55+7	70+4	1	-7,000E+01	-7,000E+01	-7,000E+01
BLEND	114+66	163+84		-3,081E+01	-3,081E+01	
ADLITTLE	97+79	156+91		2,255E+05	2,255E+05	
SHARE2B	175+115	218+90		-4,157E+02	-4,157E+02	
SC105	127+13	150+27	13	-5,220E+01	-5,220E+01	-5,220E+01
STOCFOR1	147+40	212+44		-4,113E+04	-4,113E+04	
SCAGR7	189+129	284+73		-2,331E+06	-2,331E+06	

	Iterációszám			Érték		
Név	**Szimplex**	**MBU**	**CC**	**Szimplex**	**MBU**	**CC**
SHARE1B	343+291	304+88		-7,659E+04	-7,656E+04	
BEACONFD	178+56	197+37		3,359E+04	3,359E+04	
RECIPELP	211+24	213+30		-7,714E+02	-7,714E+02	
ISRAEL		562+330			-9,292E+05	
SC205	260+59	268+69	13	-5,217E+01	-5,185E+01	-5,220E+01
LOTFI	186+160	318+330		-1,910E+01	-2,311E+01	
BOEING2	335+133	426+103		-3,150E+02	-3,150E+02	
E226		932+791	2		-1,875E+01	NaN
BANDM	1094+1079	1584+422		-1,586E+02	-1,586E+02	
GROW7	418+1184	2269+442		-8,887E+07	-8,887E+07	
SCFXM1	566+844	1444+200		1,842E+04	1,842E+04	
AGG	587+70	696+34		-3,599E+07	-3,599E+07	
SCTAP1	553+497	738+284		1,412E+03	1,412E+03	
SCAGR25	738+1722	1185+270		-1,475E+07	-1,475E+07	
AGG2		722+160			-2,024E+07	
AGG3		750+122			1,012E+07	
SCSD1	126+8102	154+2341		8,667E+00	8,667E+00	
BOEING1	1740+1710			-3,352E+02		
FFFFF800	1624					
GROW15			33			NaN
SCFXM2	1248+2255	2736+570		3,666E+04	3,666E+04	
SCRS8	685+1374	539+924		9,043E+02	9,043E+02	
GFRD-PNC	906+368	466+1116		6,902E+06	6,902E+06	
SHIP04S	385	430				
BNL1	5051	10245				
SCFXM3	1868					
WOOD1P			3			NaN

5.3. Nagytáblás programok

A következő táblázatban azt érzékeltetjük, hogy ha nagy táblákkal dolgozunk, akkor mennyire kevés feladatot képesek az algoritmusok megoldani. Itt is CC-vel jelöltük a Criss-Cross algoritmust.

Nagytáblának nevezzük azt az esetet, amikor az egész A mátrixot tároljuk a memóriában, és nem csak az előzőekben ismertetett QR felbontást használjuk. Ezekre az esetekre is elkészítettük az algoritmusok implementációját.

	Iterációszám			Érték		
Név	**Szimplex**	**MBU**	**CC**	**Szimplex**	**MBU**	**CC**
AFIRO	86+16	50+11	60483*	-7,139E+02	-1,535E+01	
KB2	85952*	118+17	55981*		-1,754E+03	
SC50A	29+136	85+23	73304*	-6,086E+01	-6,462E+01	
SC50B	75+11	110+27	106045*	-7,333E+01	-7,000E+01	
BLEND	52434*	317+146	38365*		-8,443E+01	

Kiválasztottuk a legkönnyebb 5 db feladatot és jól látszik, hogy a Criss-Cross algoritmus egyiket sem tudta megoldani. A * jelöli a táblázatban azt az iterációszámot amelynél a futást megszakítottuk, mert nem látszott, hogy közelebb került volna a végeredményhez.
A táblázatban az iterációszámok az első és a második fázis iterációinak a számát jelölik. Az általunk készített implementáció esetén az MBU-Szimplex algoritmus adta a meg a legtöbbre a helyes, vagy a helyest megközelítő eredményt, amelyet leellenőrizhetünk, ha valamelyik fenti táblázatban megkeressük az adott feladathoz tartozó eredményt.

6. Összefoglalás

A fenti eredményeket összefoglalva azt mondhatjuk, hogy a saját készítésű nagytáblás programok közül a Criss-Cross-al értük el a legrosszabb futási eredményeket. A legtöbb feladatot az MBU algoritmus oldotta meg, és a legtöbb feladatra is ez adta a helyes, vagy a helyeshez közeli eredményt. Ezekben az esetekben a Szimplex algoritmus kevesebb feladatot volt képes megoldani, mint az MBU, és ezek között a jó megoldások aránya is kisebb.

A saját implementációjú programok esetén a következőket tapasztaltuk. A Szimplex és az MBU algoritmus körülbelül azonos számú feladatot volt képes megoldani, míg a Criss-Cross algoritmus ezeknek csak a töredékét. A megoldás helyességének az aránya is nagyjából megegyezik, de az esetek nagytöbbségében azt lehet mondani, hogy az első fázis az MBU esetén tovább futott, mint a Szimplexnél, míg a második fázis iteráció számát illetően nem tudunk ilyen egyértelmű következtetéseket levonni.

Összességében a tapasztalataink azt mutatják, hogy az ipari implementációk (pl. CPLEX, Matlab Linprog függvénye) sokkal hatékonyabban oldják meg a tesztfeladatokat, míg a saját implementációink ennek csak egy, de mégis jelentős részét képesek megoldani.

Hivatkozások

[1] http://www.netlib.org/lp/data/ 2007. június 1.

[2] I. Adler and N. Megiddo. A simplex algorithm whose average number of steps is bounded between two quadratic functions of the smaller dimension. *Journal of the Association for Computing Machinery*, 32(4):871–895, 1985.

[3] I. Adler, N. Megiddo, and M. J. Todd. New results on the average behavior of simplex algorithms. *American Mathematical Society. Bulletin. New Series*, 11(2):378–382, 1984.

[4] D. Avis and V. Chvatal. Notes on bland's rule. *Mathematical Programming Study*, 8:24–34, 1978.

[5] Bilen F., Csizmadia Zs. és Illés T. Anstreicher-Terlaky típusú monoton szimplex algoritmusok megengedettségi feladatokra, *Alkalmazott Matematikai Lapok*, 2007.

[6] K. H. Borgwardt. Probabilistic analysis of the simplex method. In *Mathematical developments arising from linear programming (Brunswick, ME, 1988)*, volume 114 of *Contemp. Math.*, pages 21–34. Amer. Math. Soc., Providence, RI, 1990.

[7] Y. Y. Chang. Least index resolution of degeneracy in linear complementarity problems. (79-14), 1979. Technical Report 79-14, Department of Operations Research, Stanford University, Stanford, Calfornia, USA.

[8] G. B. Dantzig. *Linear programming and extensions.* Princeton University Press, Princeton, N.J., 1963.

[9] I. I. Dikin. Iterative solution of problems of linear and quadratic programming. *Doklady Akademii Nauk SSSR*, 174:747–748, 1967.

[10] Gy. Farkas. A fourier-féle mechanikai elv alkalmazásai (the applications of the mechanical principle of fourier). *Mathematikai és Természettudományi Értesítő*, 12:457–472, 1894.

[11] Gy. Farkas. A fourier-féle mechanikai elv alkalmazásainak algebrai alapjáról (on the algebric background of the applications of the mechanical principle of fourier). *Mathematikai és Fizikai Lapok*, 5:49–54, 1896.

[12] Gy. Farkas. Theorie der einfachen ungleichungen. *Journal für die Reine und Angewandte Mathematik*, 124:1–27, 1901.

[13] K. R. Frisch. *The logaritmic potential method for solving linear programming problems*. University Institute of Economics, Oslo, 1955. Memorandum.

[14] P. E. Gill, W. Murray, M. A. Saunders, J. A. Tomlin, and M. H. Wright. On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method. *Mathematical Programming*, 36(2):183–209, 1986.

[15] Gyapjas Ferenc. Lineáris algebra és geometria *Nemzeti Tankönyvkiadó* 258, 2001.

[16] P. Huard. Resolution of mathematical programming with nonlinear constraints by the method of center. In *Nonlinear programming*, pages 207–219. North Holland, Amsterdam, The Netherlands, 1967.

[17] T. Illés and K. Mészáros A New and Constructive Proof of Two Basic Results of Linear Programming. *Yugoslav Journal of Operations Research*, 11:15-30, 2001.

[18] Illés T. és Mészáros K. A Farkas lemma egy új és elemi bizonyitása. (Komlósi S. és Szántai T., szerkesztők) Új utak a magyar operációkutatásban (in memoriam Gyula Farkas), XXIII. *Magyar Operációkutatási Konferencia*, Pécs, 1997. október, pp. 1-16, 1999.

[19] T. Illés and T. Terlaky. Pivot versus interior point methods: pros and cons. *European Journal of Operational Research*, 140(2):170–190, 2002. O.R. for a united Europe (Budapest, 2000).

[20] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica. An International Journal of the János Bolyai Mathematical Society*, 4(4):373–395, 1984.

[21] V. Klee and G. J. Minty. How good is the simplex algorithm? In *Inequalities, III (Proc. Third Sympos., Univ. California, Los Angeles, Calif., 1969; dedicated to the memory of Theodore S. Motzkin)*, pages 159–175. Academic Press, New York, 1972.

[22] I. Maros. *Computational techniques of the simplex method*. International Series in Operations Research & Management Science, 61. Kluwer Academic Publishers, Boston, MA, 2003. With a foreword by András Prékopa.

[23] Bruce A. Murtagh. Advanced Linear Programming *McGraw-Hill Education*, 1981.

[24] K. G. Murty. Computational complexity of parametric linear programming. *Mathematical Programming*, 19(2):213–219, 1980.

[25] K. G. Murty. *Linear and combinatorial programming.* Robert E. Krieger Publishing Co. Inc., Melbourne, FL, 1985. Reprint of the 1976 original.

[26] J.L. Nazareth. Computer Solutions of Linear Programs *Oxford University Press*, 1988.

[27] K. Paparizzos. Pivoting rules directing the simplex method through all feasible vertices of klee-minty examples. *Opsearch*, 26(2):77–96, 1989.

[28] A. Prékopa. On the development of optimization theory. *Alkalmazott Matematikai Lapok*, 4(3-4):165–191 (1979), 1978.

[29] C. Roos. An exponential example for Terlaky's pivoting rule for the criss-cross simplex method. *Mathematical Programming*, 46(1, (Ser. A)):79–84, 1990.

[30] C. Roos, T. Terlaky, and J. P. Vial. *Interior point methods for linear optimization.* Springer, New York, 2006. Second edition of *Theory and algorithms for linear optimization* [Wiley, Chichester, 1997; MR1450094].

[31] Sövegjártó András. Numerikus Analízis I. jegyzet *http://numanal.inf.elte.hu/∼soveg/* 2007. június 1.

[32] Stoyan Gisbert, Takó Galina. Numerikus módszerek I-III. *ELTE-Typotex, Budapest*, 1996.

[33] T. Terlaky. A finite „criss-cross method" for solving linear programming problems. *Alkalmazott Matematikai Lapok*, 10(3-4):289–296, 1984.

[34] T. Terlaky. A convergent criss-cross method. *Optimization*, 16(5):683–690, 1985.

[35] T. Terlaky. A finite criss-cross method for oriented matroids. *Journal of Combinatorial Theory B*, 42(3):319–327, 1987.

[36] T. Terlaky and S. Z. Zhang. Pivot rules for linear programming: a survey on recent theoretical developments. *Annals of Operations Research*, 46/47(1-4):203–233, 1993. Degeneracy in optimization problems.

[37] M. J. Todd. Polynomial expected behavior of a pivoting algorithm for linear complementarity and linear programming problems. *Mathematical Programming*, 35(2):173–192, 1986.

[38] Z. M. Wang. A finite conformal-elimination free algorithm over oriented matroid programming. *Chinese Annals of Mathematics. Series B*, 8(1):120–125, 1987. A Chinese summary appears in Chinese Ann. Math. Ser. A **8** (1987), no. 1, 138.

[39] S. Zionts. The criss-cross method for solving linear programming problems. *Management Science*, 15(7):426–445, 1969.