LINEÁRIS PROGRAMOZÁSI PROGRAMOK TESZTELÉSE

TESTING LINEAR PROGRAMMING SOLVERS

Illés Tibor[1], Nagy Adrienn[2]

[1]*Department of Management Science, University of Strathclyde, Glasgow*

[2]*Neumann János Informatikai Kar, Budapesti Műszaki Főiskola, Budapest*

1. Bevezetés

Lineáris programozás az operációkutatás, de talán az egész matematika, legtöbbet hivatkozott és használt módszertana. A lineáris programozás gyökerei (Fourier-Motzkin eliminációs módszer, Farkas lemma, Minkowski tétel stb.) a XIX. századba nyúlnak vissza. A lineáris programozás fejlődése és alkalmazhatósága szempontjából alapvető eredmény volt Dantzig által megfogalmazott *szimplex algoritmus* 1947-es publikálása. Lineáris programozási modellek megfogalmazásáért és közgazdasági alkalmazásokért 1975-ben Kantorovich és Koopmans közgazdasági Nobel-díjat kaptak.

Napjainkban a lineáris programozási alkalmazások virágkorát éljük. Évente több száz, olyan dolgozat jelenik meg, amelyek lineáris programozási modellek sikeres, új alkalmazásairól számolnak be. A lineáris programozás sikeres alkalmazhatóságának számos feltétele van. Többek között ilyenek az egyre bővülő *matematikai ismeretek* (hatékony algoritmusok kifejlesztése, numerikus módszerek tökéletesítése), *megfelelő számítástechnikai háttér* (gyors számítógépek, kiváló számítógépes programok), rendelkezésre álló adatok, megfelelően képzett modellezésre és elemzésre képes szakértői gárda stb.

Cikkünkben, a lineáris programozási alkalmazások sikerességének két elengedhetetlen feltételével – hatékony algoritmusok (pivot- és belsőpontos módszerek) és számítógépes programok – foglalkozunk.

2. Lineáris programozás feladata és algoritmusai

A lineáris programozási feladat egy olyan feltételes szélsőérték feladat, amelynek a korlátozó feltételeit lineáris egyenletek, lineáris egyenlőtlenségek alkotják. Az optimalizálási feladat célfüggvénye is lineáris. A lineáris programozási feladatra példa az un. *standard feladat*

$$\min \mathbf{c}^T\mathbf{x}$$

$$\text{feltéve, hogy} \qquad \boldsymbol{A}\,\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq \mathbf{0},$$

ahol az $\boldsymbol{A}$ egy m *x* n-es teljes sorrangú valós mátrix és a $\mathbf{c}$ illetve $\mathbf{b}$ valós n- illetve m-dimenziós vektorok. Az $\mathbf{x}$ vektort a *döntési változók vektorának* nevezzük. Ismeretes, hogy a lineáris programozási feladatnak van egy un. *duális feladat*a is, amely esetünkben a

$$\max \mathbf{y}^T\mathbf{b}$$

$$\text{feltéve, hogy} \qquad \mathbf{y}^T\boldsymbol{A} \leq \mathbf{c}.$$

Mind a primál-, mind pedig a duál feladat megengedett megoldásainak a halmaza *poliéder*. Igazolható, hogy bármely primál- és duál megengedett megoldáspár esetén a primál feladat (min) célfüggvényértéke felsőkorlátot ad a duál feladat (max) célfüggvényértékére (gyenge dualitás tétel). Sőt, ha mind a primál-, mind pedig a duál megengedett megoldások halmaza nem üres akkor létezik primál- és duál optimális megoldáspár és az optimális megoldások esetén a célfüggvényértékek egyenlők (erős dualitás tétel).

A szimplex algoritmus a poliéder csúcsain (megengedett bázis megoldások) halad az optimális megoldás felé. Az új megoldás előállításakor a megengedettség biztosítására az un. *hányados teszt*et alkalmazzuk. A szimplex módszeren kívül több más pivot algoritmust (pl. Anstreicher és Terlaky, 1994 illetve Terlaky, 1985) is kifejlesztettek, amelyeknek számos, a szimplex módszertől eltérő, elméleti tulajdonságai vannak. Ezeknek az algoritmusoknak – a mi ismereteink szerint – hatékony számítógépes implementációjáról a szakirodalomban még nem számoltak be.

A lineáris programozási feladatokkal kapcsolatos elméletet és a szimplex módszert illetve annak variánsait számos könyv tárgyalja. Az érdeklődő olvasóknak Murty (1976) vagy Chvátal (1983) könyvét ajánljuk a szimplex módszerrel kapcsolatos ismeretek bővítésére.

A szimplex módszerről kimutatták, hogy bizonyos feladatokon *ciklizál*hat, azaz egy adott bázisból kiindulva véges sok iteráció után visszatérhet a kiindulási bázisba. Ciklizálásra un. *degenerált feladat*okon kerülhet sor. A ciklizálás elkerülésére és a pivot algoritmusok végességének biztosítására – az elméleti változatoknál – másodlagos pivotálási szabályokat (minimál index-, leggyakrabban mozgó változó szabály stb.) használnak. A másodlagos pivotálási szabályok jelentős kötöttséget jelentenek, ezért a számítógépes programokban nem alkalmazzák ezeket, helyettük elsősorban numerikus stabilitási illetve gyors konvergálási szempontokat vesznek figyelembe. A számítógépes implementációk esetén a ciklizálás elkerülését többnyire a számítások numerikus hibáiból adódó természetes perturbáció (Maros, 2005) vagy pedig kis nagyságrendű mesterséges véletlen perturbáció biztosítja. Érdemes megjegyezni, hogy a perturbáció bevezetése habár segít elkerülni a ciklizlást hiszen minden pivot során biztosít egy minimális javulást a célfüggvény értékében, szükségessé tesz egy végső „takarítást”, vagyis miután a perturbált feladatot megoldottuk, a perturbáció eltávolítása általában kis mértékű infizibilitást vezet be, melynek helyreállítása általában nem okoz gondot.

Hacsián *ellipszoid algoritmus*át 1979-ben publikálta (lásd Chvátal, 1983). Az ellipszoid algoritmus iterációinak a száma, a feladat változói számának és a racionális adatok méretének (bit hosszának) egy polinómjával korlátozható. Ezzel a hasznos elméleti tulajdonsággal, a polinomiális számítási komplexitással, a szimplex algoritmus nem rendelkezik. Azonban míg a szimplex algoritmus a gyakorlatban igen hatékony (a legtöbb implementáció esetén az iteráció szám O(m+n)) addig az ellipszoid algoritmus elméleti és gyakorlati lépésszáma a tapasztalat szerint igen közel áll egymáshoz ($O(n^8)$), így az ellipszoid módszer nagy méretű feladatok megoldására nem használható.

Karmarkar 1984-ben közölte a projektív skálázású belsőpontos algoritmusát. Karmarkar projektív skálázású algoritmusa is polinomiális számítási komplexitású módszer. Karmarkar eredményét követő két évtizedben több ezer publikáció jelent meg a lineáris és konvex optimalizálás területén, forradalmian átalakítva a lineáris és konvex optimalizálási feladatok megoldásának módszertanát és elméletét is egyaránt. A lineáris optimalizálás belsőpontos módszereiről egy részletes összefoglaló jelent meg az *Informatikai Algoritmusok II.* kötetében (Illés et al., 2005). A professzionális optimalizálási szoftverekben általában primál-

duál útkövető algoritmusok variánsait implementálták. Jelenleg – általában – ezek a belsőpontos implementációk jelentik a lineáris programozási feladatok leghatékonyabb megoldási módját, mert elméleti polinomiális számítási komplexitásukkal összhangban a gyakorlatban is nagyon kevés iterációra van szükségük a feladatok megoldásához, sőt általánosságban elmondható hogy a tipikus iteráció szám nem nagyobb mint 200. Érdemes megjegyezni, hogy mindezek ellenére a belső pontos eljárások közel sem egyeduralkodók. Számos esetben amikor a feladatok módosítása után gyors újra indításra van szükség, a szimplex algoritmus variánsai máig a leghatékonyabbak (pl. új feltétel bevezetése esetén [vágás feltétel az egészértékű programozási feladatoknál] vagy akár a korlátozás és szétválasztás módszerére). A duál szimplex módszerrel hatékonyan lehet kezelni az új feltételeket, míg a primál szimplex algoritmussal az új változó felvétele esetén lehet gyorsan folytatni a számításokat.

A pivot- és belsőpontos algoritmusok tulajdonságainak az elméleti és számítógépes implementációinak a gyakorlati szempontok alapján történő összehasonlításáról szól Illés és Terlaky (2002) cikke.

3. Lineáris programozási szoftverek és tesztfeladatok

A lineáris programozási megoldóknak minőségileg több fajtája létezik. Ezek közül mi a következőket vizsgáltuk:

- Professzionális szoftverek (CPLEX ver 11.0 illetve XPRESS-MP ver 2007B)
- GNU-licenszű szoftverek (LPSOLVE ver 5.5 illetve GLPK ver 4.9)

A CPLEX Optimization (mára az ILOG része) 1988-ban az 1.0-s verzióval a primál szimplex módszer implementálásaként jelent meg. A CPLEX hatékonyan képes megoldani lineáris és kvadratikus programozási feladatokat. Tartalmazza az összes jelentős gyakorlati algoritmust a primál, a duál és a hálózati szimplex algoritmuson keresztül akár kvadratikus feltételes feladatot is hatékonyan megoldó belsőpontos implementációig. Ezen eljárásokra épülve képes egészértékű feladatokat is nagyon hatékonyan megoldani. Tartalmaz ún. callable library-t is, a C, C++, C#, Java, Visual Basic és FORTRAN nyelvekhez. 2007 októberében jelent meg a 11.0-s verzió. A CPLEX programcsomagot a neumann.cs.elte.hu internet című gépen használtuk. A számítógép meghatározó paraméterei a következők: AMD Athlon, 1050 MHz processzor; 1 Gbyte, 200 MHz memória; 64 Kbyte L1 és 256 Kbyte L2 cache; Debian 3.0 kernelverzió: 2.4.28.

A Dash Optimization (mára a Fair Isaac Co. része) a világ vezető modellező és optimalizáló szoftver gyártó cége. Az XPRESS-MP egy matematikai modellező és optimalizáló rendszer, amely egyaránt tartalmazza az összes gyakorlatban bizonyított lineáris optimalizálási eljárás implementációját, képes megoldani lineáris, egészértékű, kvadratikus, nemlineáris, és sztochasztikus programozási problémákat egyaránt. Tartalmaz ún. callable library-t is, a C, C++, Java, Visual Basic nyelvekhez. 2007 decemberében jelent meg a 2007B verzió. Az XPRESS-MP programcsomagot a lime.cs.elte.hu internet című gépen használtuk. A számítógép meghatározó paraméterei a következők: 2db Dual Core AMD Opteron(tm) Processor 2210.213 MHz; 16B memória; 1024 KB cache; OpenSuse 10.1-es operációs rendszer; 2.6.16-os SMP Linux kernel.

A GLPK a GNU Linear Programming Kit rövidítése. Célja nagyméretű lineáris és kevert egészértékű programozási feladatok megoldása a módosított szimplex módszer segítségével.

A GLPK-nak is van ún. callable library része. Az utolsó 4.9-es Windows alá írt verzió 2006 márciusában látott napvilágot.

Az LPSOLVE programot nagyméretű lineáris és kevert egészértékű programozási feladatok megoldására fejlesztették ki. A program első verzióját Michael Berkelaar készítette. Az LPSOLVE-hoz is létezik ún. callable library, amelyet, C, Visual Basic, Delphi, Java programozási nyelvekből is használni lehet. Az egyes függvényei a MATLAB-bal is kompatibilisek. Jelenleg az 5.5-ös verzió a legfrissebb elérhető változata.

A fenti két GNU-licenszű megoldót a következő paraméterekkel rendelkező gépen futtattuk: AMD Athlon(tm) 64 Processor 3000+ 1,84GHz; 2,5 Gbyte, 366 MHz memória; Windows XP Professional operációs rendszer; Uniprocessor Free(32-bit) kernel; 64 KByte L1, és 512 KByte L2 cache.

A különböző gépek használatának egyszerű gyakorlati oka volt: a professzionális szoftverek licencei az említett számítógépre voltak meg. A két platform közötti generációs eltérés sajnos lehetetlenné teszi a professzionális megoldók által kapott időeredmények összehasonlítását.

A fent említett rendszereken kívül természetesen számos egyéb is létezik. Említést érdemel a Mosek mely elsősorban a belsőpontos módszerek területén erős, illetve az ingyenes és az IBM által támogatott Coin-Or solver. Ennek az érdekessége, hogy ez tartalmazza talán az egyetlen hatékony implementációjat az úgynevezett volumetric algoritmusnak.

A lineáris optimalizálási szoftverek összehasonlítására a NETLIB adatbázis tesztfeladatait használtuk és a cikkünkben ebből mutatunk be tíz-tíz futási eredményt. A cikkekben ismertetett lineáris programozási feladatok között mindenféle méretű és típusú megtalálható. A NETLIB adatbázist kifejezetten olyan célból hozták létre, hogy ismert és kevésbé ismert példákat gyűjtsenek össze általános tesztelési célokra. A NETLIB adatbázisa 98 darab lineáris programozási feladatot tartalmaz. A következő táblázatban összefoglaljuk a tíz kiválasztott lineáris programozási feladat adatait. Minden feladathoz megadjuk, hogy hány sora, oszlopa, nem nulla eleme van, és a célfüggvényének az optimumát is, valamint a kitöltöttségi arányt. A tesztfeladatok választásánál számos szempontot figyelembe vettünk. Habár a NETLIB feladatai kis méretükből adódóan mára igen egyszerű feladatoknak számítanak, azok széles körben való elterjedtsége és ismertsége biztosítja, hogy az összes megoldó fejlesztő gárdája rendelkezik azzal így a rajtuk szerzett tapasztalatot jó eséllyel a megoldókba már beépítették. Ez minimalizálja annak esélyét, hogy az amúgy igen kicsi tesztfeladat halmaz jelentős véletlen torzítást okozzon.

1. Tesztfeladatok bemutatása

Név	Sor	Oszlop	Nem nulla	Kitöltöttség	Optimum
SCTAP3	1481	2480	10734	0,292%	1.4240000000E+03
DEGEN3	1504	1818	26230	0,959%	-9.8729400000E+02
CYCLE	1904	2857	21322	0,392%	-5.2263930249E+00
STOCFOR2	2158	2031	9492	0,217%	-3.9024408538E+04
80BAU3B	2263	9799	29063	0,131%	9.8723216072E+05
BNL2	2325	3489	16124	0,199%	1.8112365404E+03
GREENBEA	2393	5405	31449	0,243%	-7.2462405908E+07

GREENBEB	2393	5405	31449	0,243%	-4.3021476065E+06
MAROS-R7	3137	9408	151120	0,512%	1.4971851665E+06
STOCFOR3	16676	15695	74004	0,028%	-3.9976661576E+04

4. Számítási eredmények

2. CPLEX és XPRESS-MP összehasonlítása

Név	Idő		Iteráció		
	CPLEX	**XPRESS-MP**	**CPLEX**		**XPRESS-MP**
SCTAP3	0.06 sec	0 sec	*335*	746	669
DEGEN3	5.83 sec	1 sec	*6508*	8108	4854
CYCLE	0.11 sec	0 sec	*0*	874	1958
STOCFOR2	0.15 sec	0 sec	*405*	1052	1188
80BAU3B	0.97 sec	0 sec	*1689*	8808	10367
BNL2	0.96 sec	0 sec	*2880*	4520	3727
GREENBEA	1.71 sec	1 sec	*1810*	6129	7514
GREENBEB	1.79 sec	0 sec	*1718*	5419	5612
MAROS-R7	8.89 sec	5 sec	*754*	3566	3952
STOCFOR3	6.46 sec	1 sec	*3017*	9833	9107

A 2. táblázatban összefoglaltuk, hogy az egyes feladatokat a CPLEX és az XPRESS-MP hány iterációban oldotta meg. A CPLEX esetében a bal oldali oszlopban az első fázis iterációinak, míg a jobb oldali oszlopban az összes iteráció számát adtuk meg.

Az iteráció számot illetően tíz esetből háromszor a CPLEX, és hétszer az XPRESS-MP oldotta meg több iterációban a feladatot. Az iteráció számok eltérésének számos oka lehet. Nyilvánvaló módon, a megoldók fő célja a megoldáshoz szükséges idő minimalizálása. A szimplex módszerek esetén annak legalább két fő lehetséges módja van, melyek nagyon eltérő iterációszámokhoz vezethetnek. Egyfelől lehetséges a megfontolt, nagyokat lépve haladás (legmeredekebb csökkenési irány vagy annak approximációi mint pl. a Devex módszer, vagy a sok kis lépést, de azt nagyon gyorsan részleges oszlopkiválasztással, vagy akár a bázis inverz méretének növekedését is figyelembe vevő részleges oszlop kiválasztással). A tényleges implementáció során általában az ezen módszerek közötti dinamikus váltások a jellemzőek.

A következő táblázatban bemutatjuk, hogy az egyes feladatokat a GLPK és az LPSOLVE mennyi idő alatt, illetve hány iterációban oldotta meg.

3. LPSOLVE és GLPK összehasonlítása

	Idő		Iteráció	
Név	**LPSOLVE**	**GLPK**	**LPSOLVE**	**GLPK**
SCTAP3	1.203 sec	0.2 sec	1059	777
DEGEN3	> 4242sec	1.6 sec	-	2006
CYCLE	2.922 sec	0.2 sec	1599	503
STOCFOR2	2.390 sec	0.2 sec	2048	584
80BAU3B	22.063 sec	3.7 sec	6140	5349
BNL2	3.141 sec	1.6 sec	2029	2064
GREENBEA	-	4.5 sec	12247	4226
GREENBEB	-	2.7 sec	4354	2657
MAROS-R7	46.484 sec	15.5 sec	4775	5402
STOCFOR3	233.625 sec	18.1 sec	17148	4761

Ezt a két softwaret azonos platformon futtattuk, így az idő összehasonlítása is lehetséges. Időben az GLPK bizonyult a gyorsabbnak, minden egyes futás során. Az iteráció számot illetően a 10 esetből kétszer a GLPK, és ötször az LPSOLVE oldotta meg több iterációban a feladatot. Találtunk viszont két olyan feladatot, amelyet az LPSOLVE nem tudott megoldani, illetve egyet, amelyben nem állt le. (A DEGEN3 feladaton valószínűleg az LPSOLVE ciklizál, ezért 70 perc után – 4242 mp után – leállítottuk a futását.) A GREENBEA és a GREENBEB feladat esetén valószínűleg numerikus hibák miatt nem talált megoldást az LPSOLVE. Az eredmények felettébb meglepőek. A két megoldót, a GLPK-t és az LPSOLVE-t általában fej-fej mellett említik, sőt számos esetben az LPSOLVE-t szokták előbbre rangsorolni. Az a tény azonban, hogy a talán legközismertebb tesztfeladaton az LPSOLVE ennyire csúfos vereséget szenved elgondolkodtató, és ezért mi egyértelműen a GLPK mellé rakjuk le voksunkat.

A következő táblázatban összevetjük, hogy az egyes megoldók hány iterációt végeztek, miközben megoldották az általunk kiválasztott tesztfeladatokat. Jelen esetben a CPLEX esetében az első oszlop tartalmazza az első fázis iterációinak a számát, és a második oszlop a feladat megoldása során elvégzett összes iteráció számot.

4. Iteráció szám szerinti összehasonlítás

Név	**CPLEX**		**XPRESS-MP**	**LPSOLVE**	**GLPK**
SCTAP3	*335*	746	669	1059	777
DEGEN3	*6508*	8108	4854	-	2006
CYCLE	*0*	874	1958	1599	503
STOCFOR2	*405*	1052	1188	2048	584

80BAU3B	*1689*	8808	10367	6140	5349
BNL2	*2880*	4520	3727	2029	2064
GREENBEA	*1810*	6129	7514	12247	4226
GREENBEB	*1718*	5419	5612	4354	2657
MAROS-R7	*754*	3566	3952	4775	5402
STOCFOR3	*3017*	9833	9107	17148	4761

Láthatjuk, hogy a CPLEX és az XPRESS-MP nagyon gyors, illetve, hogy a GNU licenszű megoldók közül a GLPK sokkal hatékonyabban oldja meg a feladatokat, mint az LPSOLVE. A bemutatott teszt feladatokon a GLPK felveszi a versenyt a professzionális megoldókkal is. A GLPK „piaci" sikerességét természetesen korlátozza az, hogy nincsen mögötte egy állandó fejlesztői-tanácsadói csapat, illetve, sajnos igen messze van egy teljeskörű megoldást nyújtó rendszertől: nincs belsőpontos eljárás beépítve így nincs például kvadratikus programozási támogatás, illetve az egészértékű megoldója nagyon szolid teljesítményt nyújt.

Ezek a megoldók jellemzően a numerikus stabilitást és a gyors számítást, a hagyományos ciklizálás elleni szabályok elé helyezik. A végesség biztosítására és a ciklizálás elkerülésére a lineáris programozás elméletében többféle, másodlagos index választási szabályt (Illés és Mészáros, 1999) vezettek be. Ezek közül a legismertebbek a Minimál index (Bland) szabály, a Most Often Selected Variables (MOSV), és a Last-In-First-Out (LIFO) index választási szabályok.

5. Összefoglalás, további kutatási kérdések

A NETLIB tesztfeladatai a komolyabb lineáris programozási megoldóknak (CPLEX, XPRESS-MP, GLPK) nem okoznak gondot. Az LPSOLVE is egészen jó iterációs számot ad, ha képes megoldani a feladatot. Sajnos a véletlenszerűen kiválasztott 10 NETLIB feladatból, hármat nem tudott megoldani, így a vizsgált két ingyenes szolver közül a GLPK-t találtuk mind gyorsabbnak mind megbízhatóbbnak.

Tekintettel arra, hogy a számítógépes implementációkban a klasszikus ciklizálás elleni technikákat (minimál index szabály, lexikografikus szimplex módszer (Chavátal, 1983)) nem alkalmazzák, mert az egyértelmű pivot választási szabály időnként olyan pivot elemet jelölhet ki, amelyik numerikus problémákhoz vezet, ezért a ciklizálás lehetősége habár igen ritka, de a jelenlegi megoldók esetén fennáll és általában (nagyon) hosszú futásokban jelentkezik, hiszen a ciklizálás tényét az implementációk általában nem képesek detektálni (ha képesek lennének nem ciklizálnának). A hosszú futás abból adódik, hogy a program észleli, hogy a célfüggvény változásával gond van és ilyenkor különböző pivot elem választási eljárásokat próbál alkalmazni. Jelentős méretű degenerált részfeladat esetén, sokszor a program kénytelen, nagyon sok egymást követő, degenerált iterációt végrehajtani, mire sikerül egy-egy nem degenerált lépést tennie.

Nincsen arról információnk, hogy a szimplex módszer alapelveitől (megengedettség fenntartása, hányados teszt alkalmazása) jelentősen eltérő típusú pivot algoritmusok (criss-cross módszer (Terlaky, 1985), monoton szimplex módszer (Anstreicher és Terlaky, 1994)) számítógépes implementációjára sor került volna. Másfelől, a szakirodalomban annak

sincsen nyoma, hogy a MOSV illetve a LIFO típusú, a minimál index szabálynál kevésbé kötött, ciklizálás elkerülését biztosító másodlagos pivotálási szabályok elterjedtek volna.

A MATLAB egy olyan fejlesztői környezetet biztosít, amely alkalmas a szimplex módszertől eltérő tulajdonságú pivot algoritmusoknak az implementációjára és azok összehasonlítására. Egy ilyen numerikus kutatásban célszerű a szimplex módszert magát is megvalósítani referencia céljából. A MATLAB implementációkkal szemben támasztott követelmények a következők lehetnek: (i) ne ciklizáljanak, (ii) oldják meg a NETLIB tesztfeladatokat, (iii) az iteráció számuk ne legyen jelentősen rosszabb, mint a GNU licenszű megoldóké (pl. LPSOLVE), (iv) legalább egy olyan ipari feladatosztályra működjenek hatékonyan, amelyik esetén a feladatok numerikus megoldása nem teljesen triviális.

5. Saját implementációinkkal megoldott feladatok

Név	Sor	Oszlop	Nem nulla	Kitöltöttség	Optimum
AFIRO	28	32	88	9,821%	-4.6475314286E+02
KB2	44	41	291	16,131%	-1.7499001299E+03
SC50A	51	48	131	5,351%	-6.4575077059E+01
SC50B	51	48	119	4,861%	-7.0000000000E+01
ADLITTLE	57	97	465	8,410%	2.2549496316E+05
BLEND	75	83	521	8,369%	-3.0812149846E+01
SCSD1	78	760	3148	5,310%	8.6666666743E+00
RECIPE	92	180	752	4,541%	-2.6661600000E+02
SHARE2B	97	79	730	9,526%	-4.1573224074E+02
SC105	106	103	281	2,574%	-5.2202061212E+01
STAIR	357	467	3857	2,313%	-2.5126695119E+02
SCAGR25	472	500	2029	0,859%	-1.4753433061E+07
AGG	489	163	2541	3,188%	-3.5991767287E+07
AGG2	517	302	4515	2,892%	-2.0239252356E+07
AGG3	517	302	4531	2,902%	1.0312115935E+07

Az előbbiekben megfogalmazott elvárásoknak eleget tevő – alapvetően elméleti jellegű – implementációk első részeredményeiről számolhatunk be. Elkészültek az első szimplex-, monoton szimplex- és criss-cross algoritmus implementációink. A ciklizálás elkerülésére a minimál index, MOSV és LIFO másodlagos pivotálási szabályokat használjuk és az így elkészült 9 lineáris programozási megoldót a NETLIB feladatokon teszteltük. A kisméretű tesztfeladatok esetén a szimplex- és monoton szimplex algoritmus variánsaink nagyon jó eredményeket mutatnak. Ezeken a kisebb NETLIB feladatokon az (i) és (iii) feltételeket kielégítik a MATLAB implementációink. A közepes méretű feladatokat is könnyedén megoldják a szimplex- és monoton szimplex algoritmusaink, de itt az iteráció számuk már elmarad az LPSOLVE iteráció számánál is, ami azonban nem meglepő, hiszen a

gyakorlatban amennyiben nem ciklizálnak (és ez meglehetősen ritka) addig még a legegyszerűbb mohó indexválasztási szabályok (pl. válasszuk a legnegatívabb redukált költségű változót) is lényegesebben hatékonyak mint a kombinatorikus jellegű indexválasztási szabályok. Nagyméretű tesztfeladatokat, főleg a flexibilisebb MOSV és LIFO szabályok alkalmazásával tudjuk megoldani, melyek egyszerre biztosítják a végességet, és képesek egyéb szempontokat is figyelembe venni, ezért ezeknek az implementációknak a továbbfejlesztésével nyerhetők a legjobb pivot alapú MATLAB programok.

A 6. táblázatban bemutatjuk az általunk készített algoritmusok iteráció szám szerinti összehasonlítását. Feltüntettük az LPSOLVE megoldási idejét is a táblázat utolsó oszlopában. A 6. és a 7. táblázatban Sz jelöli az általunk fejlesztett szimplex, és MBU az monoton bulid up szimplex algoritmust.

6. Iteráció szám összehasonlítása saját szimplex és MBU-szimplex implementációnk esetén

	LIFO		**MinInd**		**MOSV**		**LPSOLVE**
Név	**Sz**	**MBU**	**Sz**	**MBU**	**Sz**	**MBU**	
AFIRO	87	72	29	41	81	68	20
KB2	232	254	156	130	242	432	51
SC50A	75	74	60	76	70	72	46
SC50B	58	59	64	68	60	59	52
ADLITTLE	466	486	186	268	404	550	85
BLEND	523	320	175	340	603	281	96
SCSD1	3891	3249	W	951	4851	C2	91
RECIPE	385	270	233	244	589	339	55
SHARE2B	550	514	329	330	552	C2	132
SC105	415	280	138	179	560	448	100
STAIR	W	W	W	W	W	W	474
SCAGR25	3527	1761	L	1464	2928	1765	775
AGG	1275	858	697	741	1295	C1	104
AGG2	1138	C1	L	795	1238	1590	167
AGG3	1248	1569	L	874	1370	L	173

A következő jelöléseket alkalmaztuk a táblázatainkban: *W* jelöli a MATLAB által adott hibaüzenetet (Matrix is close to singular or badly scaled.), amely esetén megszakítottuk a program futását. Az L-el jelölt esetekben valamilyen hiba lépett fel a program futása során, ezért nem tudtuk megoldani a feladatot. Végül *C1*, amikor egy adott bázissorozaton megy végig, de nem változik az optimum érték, *C2* pedig azt a fajta ciklizálást, amely esetben egy adott helyről nem tud kilépni. Mindkét ciklizálás (hiszen elméletben nem lehetséges) a numerikus hibára vezethető vissza. Általános tapasztalatként elmondhatjuk, hogy a

kombinatorikus indexválasztási szabályok fő hátránya a rendkívüli függés a numerikus hibától, konkrétan hogy egy változó értéke nulla vagy sem. Mivel ezen esetekben semmilyen módon sem próbálunk nagy előrejutást elérni, csupán az elméleti módszert követjük, így a szokásos epszilon toleranciák bevezetése sem segít mérvadó módon (csupán átskálázzák a kérdést).

A 7. táblázatban bemutatjuk az általunk készített programok mennyi idő alatt oldották meg a feladatokat. A táblázatbeli adatok másodpercben értendőek. Viszonyításképpen feltüntettük az LPSOLVE megoldási idejét is.

7. Idő szám összehasonlítása saját szimplex és MBU-szimplex implementációnk esetén

Név	LIFO		MinInd		MOSV		LPSOLVE
	Sz	MBU	Sz	MBU	Sz	MBU	
AFIRO	1.422	2.11	1.750	1.672	1.906	1.922	0.031
KB2	3.438	3.312	2.797	3.422	3.219	4.234	0.047
SC50A	2.328	2.500	3.046	2.141	3.281	3.047	0.079
SC50B	2.875	2.781	3.157	2.781	3.156	2.641	0.047
ADLITTLE	4.782	5.578	3.703	4.000	4.390	5.875	0.063
BLEND	7.500	6.062	4.218	6.031	8.578	5.656	0.046
SCSD1	60.156	60.610		22.297	72.703		0.125
RECIPE	35.50	31.953	24.047	28.86	53.250	38.297	0.078
SHARE2B	11.782	13.157	8.328	8.875	12.328		0.078
SC105	12.453	11.235	6.937	8.609	15.734	14.641	0.063
STAIR							0.25
SCAGR25	3512.34	2036.45		1724.45	2787.08	2043.97	0.34
AGG	1390.74	1106.69	781.19	955.89	1424.50		0.06
AGG2	1495.52			1278.13	1602.75	2405.63	0.11
AGG3	1627.47	2347.42		1378.38	1763.61		0.09

Sajnos a criss-cross algoritmus számítógépes megvalósításával eddig még nem értünk el jelentősebb sikereket. Minden jel szerint ez az algoritmus nagyon érzékeny a rosszul kondícionált bázisokra és ilyenek után valószínűtlenül hosszú futásokat produkál. Jobb implementáció érdekében sok numerikus ellenőrzést kell beépítenünk a kódunkba. A minimál index szabály használata csak még súlyosbítja a helyzetet, mert néhány feladat esetén igen gyorsan jelöl ki numerikus szempontból nem megfelelő pivot elemet. A criss-cross algoritmus tesztelése során szerzett általános negatív tapasztalatok ellenére, néhány igen érdekes és elgondolkodtatóan pozitív számítási eredményről is beszámolhatunk: azokat a tesztfeladatokat, amelyeket a criss-cross algoritmus valamelyik másodlagos pivot elem választási szabály alkalmazásával megoldott, minden esetben jelentősen kevesebb iterációval oldotta meg, mint a másik két algoritmus leggyorsabb változata.

A 7. táblázatban bemutatott számok megértéséhez fontos néhány megjegyzést tenni. A nagy számok elsősorban a bázis inverzének kiszámítására használt nem hatékony eljárásra vezethetőek vissza, mely esetünkben a MATLAB QR felbontása és annak MATLAB általi újraszámítása volt. Ritkás LU felbontással és annak újraszámításával nagyságrendű javulást lehetne elérni. Érdekes módon, a MATLAB nem támogatja az LU felbontás újraszámítását (vélhetően azért, mert a legtöbb LU update eljárás nem közvetlenül a felbontást módisítja, csak annak egy részét és segéd transzformációs mátrixokat [melyeket gyakran ETA fileoknak neveznek] vezet be (Nazareth, 1987)). Egy jó LU felbontást adó MATLAB függvény implementálása, jelentős javulást eredményezhetne a MATLAB implementációink numerikus stabilitásában.

A 8. táblázatban bemutatjuk a fenti feladatokon, az általunk készített criss-cross algoritmussal elért eredményeinket. Ebben a táblázatban LPS jelöli az LPSOLVE megoldási idejét és iteráció számát.

8. Idő és iteráció szám összehasonlítása saját criss-cross implementációnk esetén

Név	Idő				Iteráció			
	LIFO	MinInd	MOSV	LPS	LIFO	MinInd	MOSV	LPS
AFIRO	0.547	C1	0.546	0.031	35		48	20
KB2	1.360	W	5.625	0.047	264		349	51
SC50A	0.735	0.734	0.641	0.079	5	5	5	46
SC50B	0.734	0.829	0.813	0.047	1	1	1	52
ADLITTLE	11.297	C2	C2	0.063	2113			85
BLEND	5.125	W	C1	0.046	502			96
SCSD1	C1	C2	C1	0.125				91
RECIPE	4.969	W	5.250	0.078	3		30	55
SHARE2B	22.547	W	C1	0.078	1474			132
SC105	2.031	2.031	2.031	0.063	13	13	13	100
STAIR	8190.109	C2	C1	0.25	16731			474
SCAGR25	6326.031	C1	C2	0.34	6420			775
AGG	W	W	W	0.06				104
AGG2	W	W	W	0.11				167
AGG3	W	W	W	0.09				173

Reméljük, hogy a közeljövőben olyan MATLAB implementációkat tudunk előállítani, amelyek az (i)-(iii) feltételeknek eleget tesz.

Köszönetnyilvánítás: Illés Tibor köszönetet mond az OTKA támogatásának, amelyet a T 049789 számú kutatási pályázat keretében nyújtottak. Továbbá köszönetet mondunk Csizmadia Zsoltnak, aki a cikk korábbi változatához számos hasznos megjegyzést fűzött.

Irodalomjegyzék

[1] Anstreicher K. and Terlaky T. (1994) A monotobic build-up simplex algorithm for linear programming. Operation Research, 42, 556-561.

[2] Chvátal V. (1983) Linear programming, W. H. Freeman and Company, New York.

[3] GLPK ver 4.9. Letölthető a következő internet címekről:
http://gnuwin32.sourceforge.net/packages/glpk.htm,
http://www.gnu.org/software/glpk/

[4] Illés T. és Mészáros K. (1999) A Farkas lemma egy új és elemi bizonyítása. In: Új utak a magyar operációkutatásban. Szerkesztők: Komlósi S. és Szántai T. Dialógus Campus Kiadó, Budapest, 73-88.

[5] Illés T., Nagy M. és Terlaky T. (2005) Belsőpontos algoritmusok. In: Informatikai Algoritmusok II. Alkotó szerkesztő: Iványi A. ELTE Eötvös Kiadó, Budapest, 1230-1297.

[6] Illés T. and Terlaky T. (2002) Pivot Versus Interior Point Methods: Pros and Cons. European Journal of Operational Research 140, 170-190.

[7] ILOG CPLEX 9.0 User's Manual
Elérhető a következő cmen: http://www.ilog.com/products/cplex/

[8] LPSOLVE ver 5.5. Letölthető a következő internet címekről:
https://sourceforge.net/projects/lpsolve
http://lpsolve.sourceforge.net/

[9] Maros I. (2003) Computational techniques of the simplex method. International Series in Operation Research & Management Science 61. Kluwer Academic Publishers, Boston

[10] Murty K. (1976) Linear and combinatorial programming, John Wiley & Sons, Inc., New York.

[11] Nagy A.(2007) Új típusú pivot módszerek numerikus összehasonlítása a lineáris programozásban, diplomamunka, ELTE IK Elérhető a következő címen: http://people.inf.elte.hu/parad/Magamrol/NagyADiplomamunka.pdf

[12] Nazareth J.L. (1987) Computer Solution of Linear Programs Oxford University Press, Oxford and New York

[13] Terlaky T. (1985) Véges criss-cross módszer. Alkalmazott Matematikai Lapok.

[14] XPRESS-MP Release 11 Reference Manual
Elérhető a következő címen: http://www.dashoptimization.com/