Debreceni Egyetem

Informatikai Kar

RetroLP elnevezésű lineáris programozási programcsomag használata saját fejlesztésű alkalmazásban, C# környezetben

Témavezető:
Dr. Bajalinov Erik
(tudományos főmunkatárs)

Készítette:
Gál János Zoltán
(Programtervező Matematikus)

Debrecen
2007

1. Bevezetés

A diplomamunkám alapvető célja Gavriel Yarmish és Richard Van Slyke által készített RetroLP programcsomag felhasználása egy általam írt alkalmazásban. A RetroLP programcsomag a szimplex módszert megvalósítva LP feladatok megoldását végzi. Számos más szoftver létezik, amit felhasználhattam volna az alkalmazásom motorjaként, azonban figyelembe véve a feltételeket és a szoftverek tudását a RetroLP tűnt a legideálisabb választásnak. A főbb szempontok, amik alapján választanom kellett a következők:

- fizetős vagy ingyenes
- milyen eredményeket produkál nehéz feladatokon
- rendelkezik-e szabad forráskóddal

Az első pont általában szorosan összefügg a második ponttal, ugyanis az figyelhető meg, hogy a fizetős szoftverek motorjai, jobban teljesítenek, mint az ingyenes társai (mint ahogy azt várnánk is). Elsősorban ez az igazán nehéz és bonyolult feladatoknál ütközik ki. A harmadik pont, pedig elsősorban akkor fontos, ha tovább szeretnénk fejleszteni a motort, ami megoldja az LP feladatot. (E dolgozat fő célja nem ez volt, ezért ennek a szempontnak a figyelembevétele nem is lett volna szükséges, azonban azt is lényegesnek tartottam, hogy később bárki javítani tudja a motor működését). A szóba jöhető fizetős motorok (Lindo, CPLEX, LGO, Conopt) elég drágák, így ezek az első körben kiestek. Ezen fizetős motoroknak ugyan léteznek szabadon felhasználható ingyenes változatai, azonban igen komoly korlátokkal, amely elsősorban a változók és megszorítások számaira vonatkozik. Ennek következtében egy komolyabb probléma megoldására nem alkalmasak, így ezek felhasználását is kizártam. Az ingyenes kategóriában is számos LP feladat megoldását megvalósító szoftver található, mint például a CoinMP, GLPK vagy a LPSolve. Ezeknél már nincsenek korlátok a változók és megszorítások számaira, amely nagy előny. A negatívum, ami megemlíthető a nevük mellettük, azaz hogy kevésbé megbízhatóak a nehezebb problémáknál, és a többségük forráskódja nem nyilvános. A keresés közben akadtam a már említett RetroLP programcsomagra, amelyet a brooklyni professzor Richard Van Slyke és PHD hallgatója Gavriel Yarmish fejlesztett. A programcsomag fejlesztése még 2001-ben kezdődött C nyelven, de a későbbiekben már számos javítás és fejlesztés C++ nyelven történt.

A program GNU-GPL (General Public License) licenc hatálya alá tartozik, amely értelmében szabadon felhasználható, és továbbfejleszthető a kód, azzal a kikötéssel, hogy az eredeti, és a módosított kódrészletekben is szerepeltetni kell egy hivatkozást a szerzőre és a hatályos licencre. Annak ellenére, hogy ingyenes szoftverről van szó igen szép eredményt produkál az igazán nehéz feladatoknál is. Az hogy a RetroLP ingyenes, nyílt forráskódú, és a nehéz feladatokon is jól működik egyértelművé tette, hogy ezt a programot kell használjam az alkalmazásom motorjaként.

A diplomamunkámban nem csak program ismertetésére és a készítése közbe felmerült problémákra térek ki, hanem ismertetem a szimplex módszert, valamint olyan technikákat, amelyek segítségével hatékonyabbá, illetve pontosabbá tehető az eljárás. Részletesen kitérek és egy konkrét példán keresztül bemutatom az MPS fájlok felépítését, amely segítségével LP feladatokat írhatunk fel szabványosan. Erre azért van szükség, mert a RetroLP is ilyen formátumban várja az LP feladatot.

2. A program és a RetroLp bemutatása

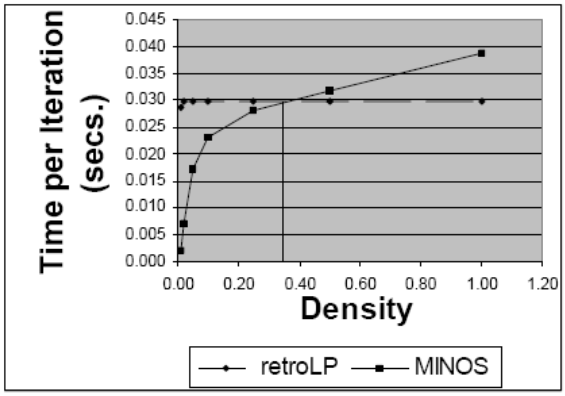
2.1 Módosított és az eredeti szimplex módszer

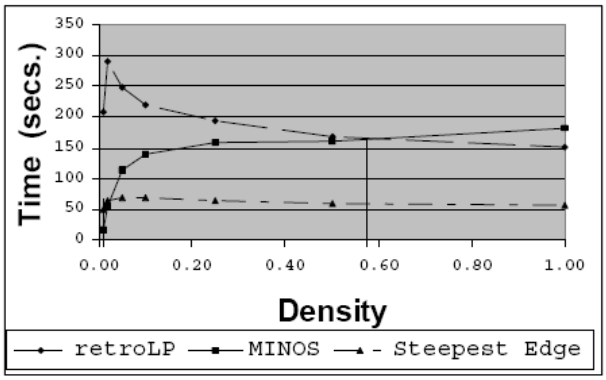
Mielőtt ismertetném az általam készített programot, szeretném bemutatni a programom motorjaként használt RetroLP programcsomagot, elsősorban előnyeit és felépítését. A RetroLP arra a George Dantzing által megalkotott LP feladatok megoldását szolgáló szimplex módszerre épül, amelyet a 20. század egyik legjelentősebb algoritmusának tartanak. A módszernek két fő változata van, az egyik az eredeti szimplex módszer, a másik pedig a módosított szimplex módszer. A RetroLP, mint ahogy azt a neve is mutatja az előbbi képviselője, azaz az eredeti szimplex módszert valósítja meg. Úgy tűnik, manapság a szimplex módszer implementációi a módosított szimplex módszerre épülnek, ugyanis ez sokkal hatékonyabb a ritka LP feladatok esetén, amely feladatok igen gyakoriak a mindennapi életben. Azonban nem szabad elfelejtkezni az eredeti szimplex módszerről sem, mivel számos előnyös tulajdonsága van. Mindenek előtt az eredeti szimplex módszer hatékony a sűrű LP feladatok esetében, amely feladatok általában nem szokványos problémáknál jelentkeznek. Továbbá az eredeti szimplex módszer könnyen és hatékonyan továbbfejleszthető, úgy hogy rosszul skálázott feladatot is megoldjon. Arról sem szabad elfeledkezni, hogy a módszer alapvető egyszerűsége lehetővé teszi a hatékonyság növelést (elsősorban itt a számítógépek belső memóriájának megfelelő használatára kell gondolni)

Az eredeti szimplex módszerben a számítások túlnyomó többségét az teszi ki, hogy a régi szimplex táblából kiszámítjuk az újat. Általában a szimplex táblában szerepelő a_{ij} és c_j elemek együtthatói minden egyes bázistranszformációt követően változnak. Ez különösen problémás, ha az oszlopok száma lényegesen nagyobb, mint a sorok száma. Ezenkívül az is problémát okoz, ha az elemek többsége nulla, mivel néhány iterációs lépést követően nullától különböző elemmé válnak, annak köszönhetően, hogy az általános szimplex módszernél szinte minden együttható értéke változik az egyes iterációk során. Ahhoz hogy végrehajtsunk egy iterációt a szimplex módszer során, csupán a szimplex tábla következő részeire van szükség (ha feltételezzük, hogy a klasszikus oszlopválasztási szabályt alkalmazzuk):

- A célfüggvény együtthatói
- A bázisba kerülő oszlop együtthatói
- A bázisban szereplő együtthatók aktuális értékei

Az első pont segítségével választjuk ki a bázisba kerülő vektort, a második és harmadik pont segítségével választjuk ki a bázisból kikerülő vektort. Tehát a teljes szimplex tábla csak egy kis részét használjuk fel (két oszlopot és egy sort).
Az 1950 –es évek elején rájöttek, hogy ez a három vektor kiszámítható egy olyan táblázatból amelynek egyik része fix és megegyezik az induló szimplex táblával, a másik része pedig egy folyamatosan változó segéd táblázat. Ennek következtében kevesebb munkát igényelnek az egyes iterációs lépések. Személetesen módosított szimplex eljárásnak nevezik, mivel a táblázat szerkezetén módosítás történt. Ez a módosított szimplex eljárás hatékonyabb ritka lineáris problémák esetén, valamint abban az esetben, ha az oszlopok száma nagyobb, mint a sorok száma.
A RetroLP –t készítő Richard Van Slyke végzett egy tesztelést a módosított és az eredeti szimplex módszer hatékonyságát illetően. A két program, amin végrehajtotta a tesztelést a RetroLP és a MINOS volt. A MINOS egy módosított szimplex módszeren alapuló programcsomag. Először úgy hasonlította össze a két programot, hogy mind a kettő a klasszikus oszlopválasztási szabályt alkalmazza. Majd pedig úgy, hogy a RetroLP a legmeredekebb él módszerét használta az oszlopválasztásnál, a MINOS pedig továbbra is a klasszikus oszlopválasztás módszerével dolgozott (ennek az oka az, hogy a MINOS nem támogatja a legmeredekebb él oszlopválasztási módszert). A lineáris feladatok, amiket a tesztelésre használt körülbelül 500 megszorítással és 1000 változóval rendelkeztek.





Az első esetben az figyelhető meg, hogy a RetroLP esetén az egyes iterációk időigénye független a sűrűségtől, míg a MINOS esetében a sűrűséggel együtt nő az iterációk időigénye. A metszéspont körülbelül 0.5 –nél található. A második esetben a teljes futási idő került mérésre. Az ábrán látható a RetroLP futási ideje klasszikus oszlopválasztási szabállyal és legmeredekebb él szabállyal is. A MINOS eredményeinek mérése klasszikus oszlopválasztási szabállyal történt. A két metszéspont 0.58 és 0.05 nél található. Az ábrákból szépen leolvasható az, hogy vannak a lineáris programozásnak olyan szegmensei, amelyek a klasszikus oszlopválasztási szabállyal gyorsabban megoldhatóak.
Az eredeti szimplex módszer általános tárgyalása után térjünk át konkrétan a RetroLP –re. A program 22 fájlból áll, amelyből 4 header fájl. Az eredeti verzió egy konzolos program, amely az input fájl nevét egy paraméterként kapja induláskor. A fájl egy MPS formátumú fájl, amely pontos felépítését később részletezem. Futás előtt lehetséges számos paramétert beállítani, mint például az iterációk számát, az oszlopválasztás módját (klasszikus, legmeredekebb él), a pontosság nagyságát, valamint hogy minimalizálásról, vagy maximalizálásról van-e szó. A paraméterek beállítását követően a program a szimplex módszert alkalmazva kiszámolja az optimális értéket és a képernyőn megjeleníti.

2.2 Program bemutatása

A program fejlesztése előtt több lehetséges út közül is lehetett választanom. Az egyik út az volt, hogy Visual C++ használok a grafikus felület, megírásához, a másik pedig az, hogy C# -ban fejlesztem a grafikus részt. Mindkét lehetőségnek megvolt a maga előnye és hátránya. Az előbbi esetben a nehézkes vizuális elemek kezelése mellett könnyen megvalósítható a RetroLP kód beágyazása, mivel az C –ben és C++ –ban íródott. Az utóbbi esetben viszont pont az ellenkezője mondható el, ugyanis C# –ban sokkal könnyebb megvalósítani a program vizuális részét, arról nem is beszélve, hogy sokkal több grafikai lehetőség is van. Ami viszont sokkal problémásabb az a RetroLP beágyazása. E két lehetőség közül az utóbbit választottam, mivel összességben sokkal egyszerűbbnek tűnt a megvalósítása. A problémát a C és a C# közti nyelvi különbségek jelentették, amelyek áthidalását úgy oldottam meg, hogy az eredeti RetroLP kódból egy dll fájlt fordítottam. A dll előállításhoz, csak úgy, mint a grafikai részhez a Microsoft Visual Studio 2005 fejlesztői környezetet használtam.

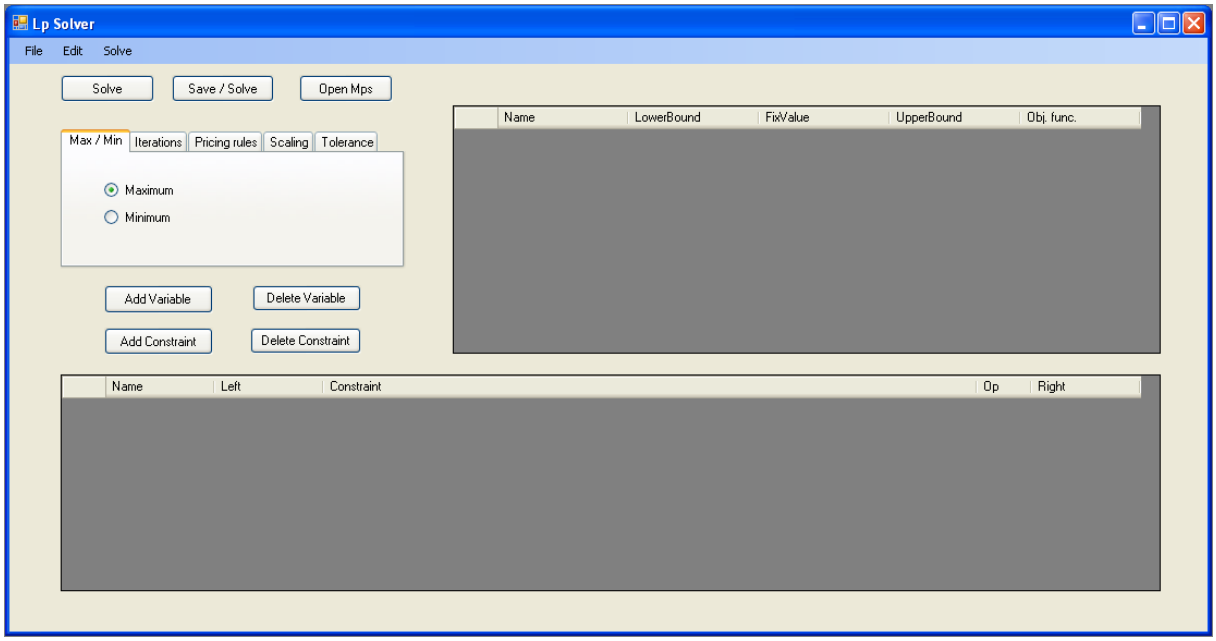
2.3 Dll létrehozás a RetroLP kódból

A cél az volt, hogy olyan dll fájlt készítsek a RetroLP kódból, amelyet később meg tudok hívni a C# –ból. Ahhoz, hogy a C# –ban meghívhassak egy dll –t először referenciaként hozzá kell rendelni a projecthez, majd példányosítani kell egy a dll –ben szereplő osztályt és ennek az osztálynak a metódusait lehet meghívni. A probléma az esetemben az volt, hogy a RetroLP –ben (ahogyan már említettem) vannak C és C++ kódok is, amelynek eredménye az, hogy a kód egyik része eljárásorientált, míg a másik része objektumorientált módon van megírva. A C# –ban, a dll –ből csak az objektumorientáltan módon megírt metódusok látszódnak, így kézenfekvő volt az a megoldás, hogy minden RetroLP –ben szereplő függvényt egy *namespace* –be és azon belül *classok* –ba csomagolok. Ennek a megoldásnak az a hátránya, hogy rengeteg linkelési hibát okoz, aminek a javítása óriási munka. Ezért egy olyan megoldást választottam, amelyet sokkal egyszerűbb és gyorsabb megvalósítani. Első lépésben átneveztem a *main* függvényt a *doit* –ra (mivel a dll –ben nem szerepelhet *main*), majd létrehoztam egy osztályt és azon belül egy metódust (`call_RetroLP`), amely segítségével meghívom a *doit* függvényt. Ezzel kvázi becsomagoltam az eredeti *main* függvényt egy a C# –ból is látható metódusba, így nagyon sok felesleges munkát kerültem el.
Megjegyezném, hogy kisebb hibák így is adódtak a fordításnál, azonban ezek javítása nem volt nehéz. Egyik ilyen probléma a *pow* függvénnyel akadt, de ezt sikerült megoldani, úgy hogy saját magam megírtam a hatványozás függvényt.
Egy másik igen jelentős problémám a paraméter átadással volt, mint azt ahogy már írtam az eredeti RetroLP futását számos beállítási lehetőséggel lehetett befolyásolni. A futási paraméterek beállításának túlnyomó többsége konzol ablakon keresztül történt, amelyre dll hívásnál nincsen lehetőség. Ezért paraméter listában átadtam a `call_RetroLP` metódusnak, majd onnan továbbadtam a *doit* (eredetileg *main*) függvénynek a futásra vonatkozó paramétereket. A *doit* elején pedig kézzel beállítottam a kívánt értékeket. (Az átadott paraméterek a következők voltak: iterációk száma, oszlopválasztás módszere, minimum vagy maximum keresése, skálázás, tolerancia nagysága) A program fejlesztése közben, ahogy egyre több paraméterrel kívántam befolyásolni a dll futását, egy érdekes jelenségre derült sor, miszerint, ha háromnál több paramétert adok át, a csomagoló metódusomnak, akkor a dll hívást követően leáll a program. Ezt kiküszöbölendő, négy paramétert egy változóba sűrítettem. Ezt azért tudtam megtenni, mivel a négy paraméter lehetséges értékei 0 és 4 közé

estek. Az ötletem az volt, hogy egy négyjegyű szám négy helyértékének feleltetem meg a négy paraméter, és `call_RetroLP` csomagoló metódusomban fogom visszaalakítani a paramétereket maradékos osztással. Ennek köszönhetően a négy plusz egy paraméter helyett elegendő volt átadni csak egy plusz egy paramétert.

2.4 Vizuális felület bemutatása

A program indítását követően a vizuális felület a következő képen néz ki:

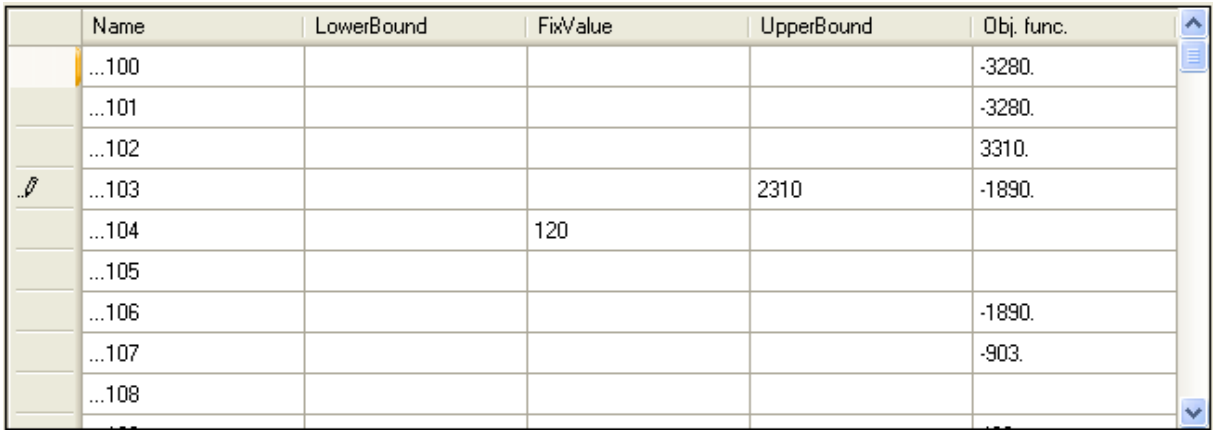


A felület 5 fontosabb részre bontható, amelyek a következők:

1. Változók megjelenítése
2. Megszorítások megjelenítése
3. Feladat módosítást, megoldását segítő gombok
4. Futási paraméterek módosítására szolgáló fülek
5. Menü

Nézzük meg sorban az egyes felületek szerkezetét és funkcionalitását.

Változók megjelenítése: A felület jobb felső sarkában található táblázat az LP feladat változóit tartalmazza a következő formában:

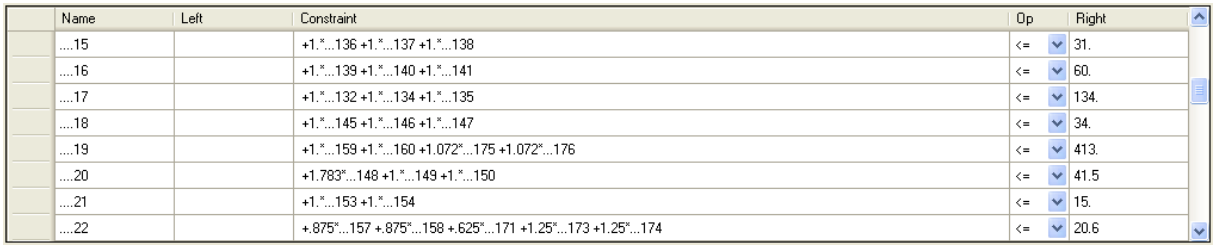


	Name	LowerBound	FixValue	UpperBound	Obj. func.
	...100				-3280.
	...101				-3280.
	...102				3310.
	...103			2310	-1890.
	...104		120		
	...105				
	...106				-1890.
	...107				-903.
	...108				

A táblázat minden egyes sora egy-egy változónak felel meg. A táblázat oszlopai az egyes változók tulajdonságait tartalmazza, amelyek a következők:

- Name: A változó nevét tartalmazza.
- LowerBound: Amennyiben üres a mező úgy a változóra a $\mathbf{x \geq 0}$ feltétel teljesül, ha pedig egy szám szerepel az adott cellában, akkor x nagyobb, vagy egyenlő, mint ez a szám.
- FixValue: Amennyiben egy változó ezen cellája nem üres, akkor a változónak kötött értéke van, mégpedig a cellában megadott érték.
- UpperBound: A változó felső korlátjának megadására szolgál. Amennyiben üres a cella úgy a változó felső korlátja a plusz végtelen, ha egy konkrét szám áll a cellában, akkor a változó értéke kisebb, vagy egyenlő, mint a megadott szám
- Obj. func.: Ebben az oszlopban az található, hogy az egyes változók milyen együtthatókkal szerepelnek a célfüggvényben.

Megszorítások megjelenítése: A felület alsó részén található táblázat az LP feladat megszorításait tartalmazza a következő formában:



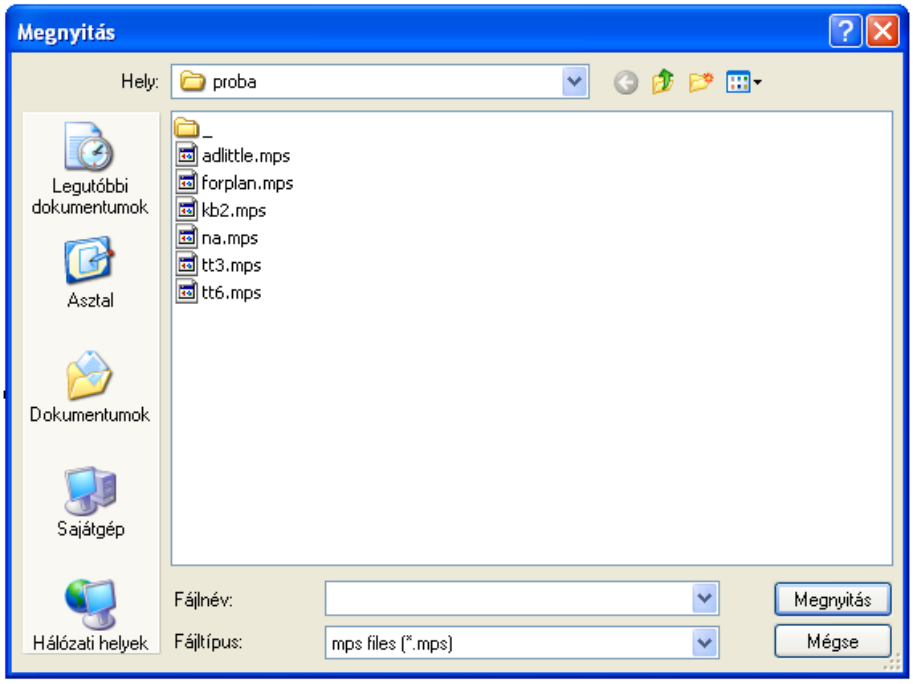
Name	Left	Constraint	Op	Right
....15		+1.*...136 +1.*...137 +1.*...138	<=	31.
....16		+1.*...139 +1.*...140 +1.*...141	<=	60.
....17		+1.*...132 +1.*...134 +1.*...135	<=	134.
....18		+1.*...145 +1.*...146 +1.*...147	<=	34.
....19		+1.*...159 +1.*...160 +1.072*...175 +1.072*...176	<=	413.
....20		+1.783*...148 +1.*...149 +1.*...150	<=	41.5
....21		+1.*...153 +1.*...154	<=	15.
....22		+.875*...157 +.875*...158 +.625*...171 +1.25*...173 +1.25*...174	<=	20.6

A táblázat minden egyes sora egy-egy megszorításnak felel meg. A táblázat oszlopai az egyes megszorítások tulajdonságait tartalmazza, amelyek a következők:

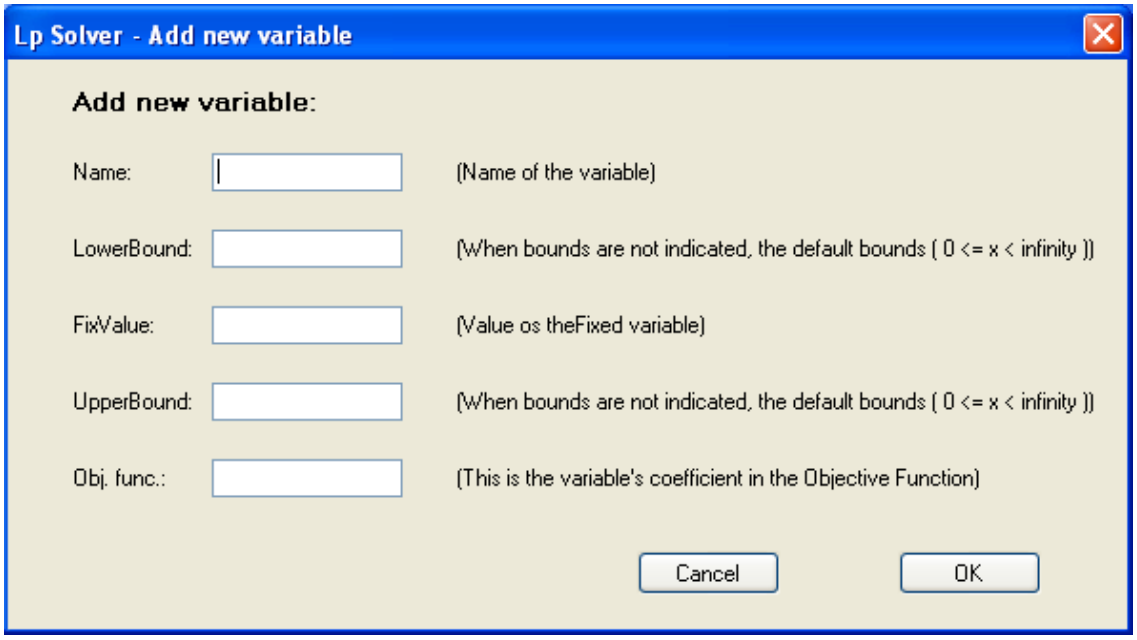
- Name: A megszorítás nevét tartalmazza
- Left: Ezzel egy intervallumot lehet megadni a megszorításhoz. Ha ≤ operátort tartalmaz a sor, akkor a megszorítás alsókorlátja a *Right* oszlopban lévő szám mínusz a *Left* oszlopba lévő szám. Ha viszont ≥ operátort tartalmaz a sor, akkor a megszorítás felső korlátja a *Right* oszlopban lévő szám plusz a *Left* oszlopba lévő szám .(Az itt megadott érték fog a RANGES szekcióban szerepelni az MPS fájlban)
- Constraint: Ebben az oszlopban szerepelnek a feltételek bal oldali, tehát a változók az együtthatóikkal. Az itt szereplő adatok struktúrájára szigorú szabályok vonatkoznak, amely a következők. Az együtthatók előtt kötelező szerepeltetni a műveleti jelet (+ vagy -, még a legelső esetben is), az együttható után közvetlenül egy * jel szerepel, majd azt követően a változó neve. Szóköz mindig csak a + és – műveleti jel előtt van (kötelező jelleggel, tehát a cella első karaktere is szóköz). Például: 1.2x1 +3*x2 –4*x3
- Op.: Ebben az oszlopban adható meg a megszorításhoz tartozó operátor. A használható operátorok a következők: **≥ , = , ≤**
- Right: A megszorítások jobb oldalát lehet ebben az oszlopban megadni, ahol üresen van hagyva ez a cella, ott automatikusan a nulla szerepel.

Feladat módosítást, megoldását segítő gombok: Ezek a gombok a felület bal felső részében találhatóak. Nézzük sorra a hét különböző gomb működését.

- Open Mps: Ennek segítségével lehet kijelölni, hogy melyik MPS fájlt akarjuk megnyitni.

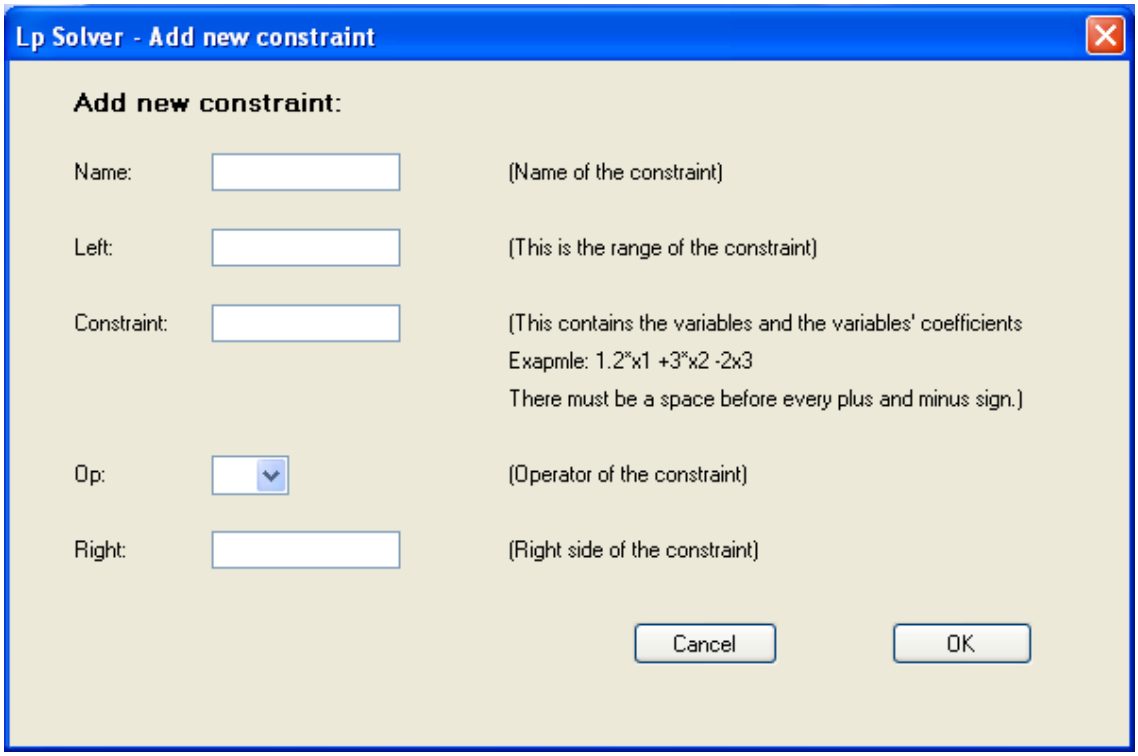


- Solve: Ennek segítségével lehet kiszámítani a problémánk optimális megoldását. A program ilyenkor hívja meg a dll fájlt.
- Save / Solve: Ez hasonlóan a Solve –hoz kiszámítja az optimális megoldást, azzal különbséggel, hogy előtte elmenti a végrehajtott módosításokat a feladaton.
- Add Variable: Ennek a gombnak a segítségével lehet új változót létrehozni. A változó tulajdonságainak a megadást a következő ablak segíti:



Az ablakban megadható adatok megegyeznek a változókat magába foglaló táblázat oszlopaival, ezért nem térek ki ismét ezek funkciójának részletezésére.

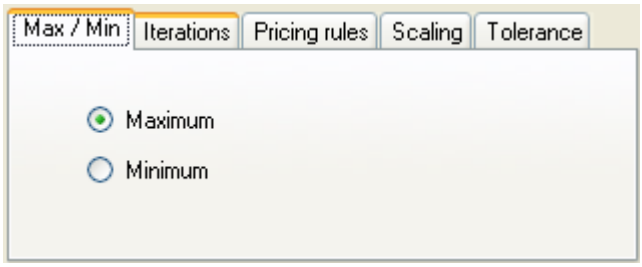
- Delete Variable: Ennek a gombnak a segítségével lehet a változók táblájából kitörölni egy sort. Ehhez ki kell jelölni a törölni kívánt sort, úgy hogy a legelső cellájára kattintunk, majd rákattintani a Delete Variable ikonra.
- Add Constraint: Ennek a gombnak a segítségével lehet új megszorításokat létrehozni. A felugró ablakban megadható adatok megegyeznek a megszorításokat magában foglaló táblázat oszlopaival, ezért itt sem térek ki még egyszer ezek részletezésére. A megszorítások tulajdonságainak a megadást a következő ablak segíti:



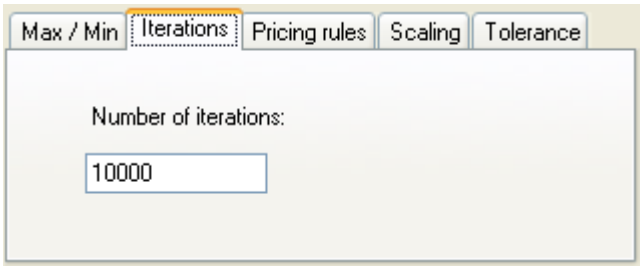
- Delete Constraint: Ennek a gombnak a segítségével lehet a megszorítások táblájából kitörölni egy sort. Ehhez ki kell jelölni a törölni kívánt sort, úgy hogy a legelső cellájára kattintunk, majd rákattintani a Delete Constraint ikonra.

Futási paraméterek módosítására szolgáló fülek: Ezen fülek segítségével lehet dll –t felparaméterezni. Az öt különböző fül más és más funkciók paramétereinek a beállítását szolgálja. Tekintsük most egyesével végig, hogy melyik fül mit szolgál.

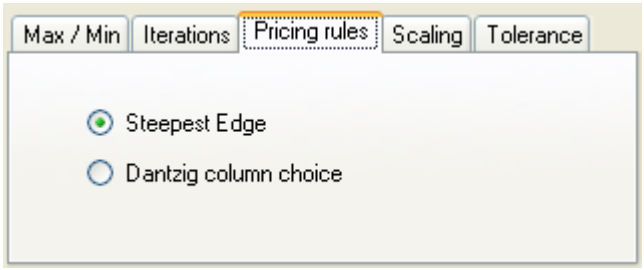
- Max/Min fül: A fülön két rádió gomb található (egy Maximum és egy Minimum), aminek a segítségével lehet megadni, hogy a célfüggvénynek a minimumát, vagy a maximumát keressük.



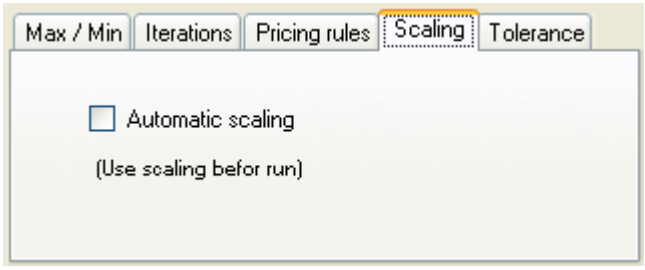
- Iterations fül: A maximális iterációk számát lehet megadni. Minél nagyobb a megadott szám, annál később tekinti úgy a dll, hogy végtelen ciklusba futott a szimplex módszer, így nő annak az esélye, hogy megoldást talál.



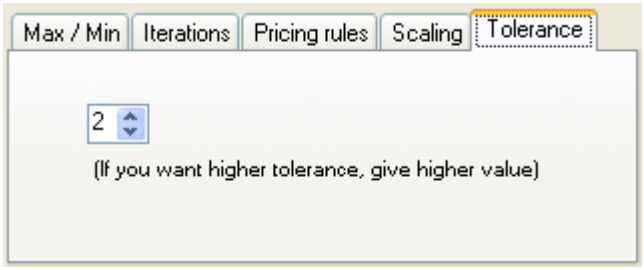
- Pricing rules fül: A bázisba kerülő oszlop kiválasztásának módját lehet itt megadni. Alap esetben a *Steepest Edge* eljárást használja, azonban helyette választható a klasszikus *Dantzing column choice* eljárás. A két eljárás közötti különbségre még visszatérek.



- Scaling fül: Amennyiben a checkBox bejelölésre kerül, úgy a szimplex módszer végrehajtása előtt az indulótáblán egy skálázás kerül végrehajtásra. Arra, hogy milyen esetekben célszerű ezt alkalmazni, és hogyan működik az eljárás, még később visszatérünk.



- Tolerance fül: A pontosság javítására szolgál. Minél nagyobb a megadott tolerancia, annál pontosabb a kapott eredmény.



Menü: A menüben azok a funkciók vannak egybegyűjtve, amelyek amúgy is elérhetőek a gombok segítségével. Tekintsük át pontosan melyek is ezek:

1) File
   a) New (Új fájl létrehozása)
   b) Open (MPS fájl megnyitása)
   c) Save (MPS fájl mentése)
   d) Exit (Ablak bezárása)
2) Edit
   a) Variable
      i) Add Variable (Új változó létrehozás)
      ii) Delete Variable (Változó törlése)
   b) Constraint
      i) Add Constraint (Új megszorítás létrehozás)
      ii) Delete Constraint (Megszorítás törlése)
3) Solve
   a) Solve (Probléma megoldás)
   b) Save / Solve (Mentés, majd a probléma megoldás)

2.5 Vizuális elemek háttere

A felületen megjelenő kontrol elemek többsége egyszerűen implementálható, valamint a számos beépített metódus segítségével könnyen módosíthatók a tulajdonságai és paraméterei. Gondolok itt elsősorban a következő elemekre: *Button*, *Label*, *TextBox*, *CheckBox*, *NumericUpDown*, *RadioButton.* Az egyszerű használhatóság miatt ezekre most külön nem térek ki. Vannak azonban ennél kicsit összetettebb kontrol elemek is a C# –ban, amelyek közül néhányat én is felhasználtam.

Az egyik ilyen kontrol elem a változókat és megszorításokat megjelenítő *DataGridView.* Ha C# –b valamilyen adatokat akarunk rendezett formában, csoportosítva megjeleníteni a képernyőn, akkor a legcélszerűbb a *DataGridView* –t használni. A *DataGridView* számos beépített lehetőséget tartalmaz, amelynek implementálásával a felhasználónak nem kell foglalkoznia. Elsősorban itt a módosításokra, kijelölésre és a megjelenítésre gondolok. A

DataGridView, mint ahogy az a nevében is benne van, adatokkal dolgozik, az adatokat pedig valamilyen formában tárolni kell. Adatok forrásaként több minden is megadható, mint például adatbázis, web servise, vagy egy osztály. Én ez utóbbit választottam és mind a változókat, mind a megszorításokat megjelenítő *DataGridView* esetében létrehoztam egy-egy osztályt, aminek pont annyi adattagja van, mint ahány oszlopa a táblázatomnak. Fontos, hogy az osztályon belül ne csak a konstruktort írjuk meg, hanem minden egyes adattaghoz hozzunk létre egy property –t, ugyanis a *DataGridView* a set és get tagfüggvények segítségével éri el az adattagokat. A property megadása a következő formában történik:

```
public String Name
        {
            get
            {
                return name;
            }
            set
            {
                name = value;

            }
        }
```

A get tagfüggvény a name adattag aktuális értékével tér vissza, míg a set tagfüggvény a name adattag értéknek a megváltoztatására szolgál. Az így létrehozott osztály egy-egy példánya tartalmazza a *DataGridView* egyes sorainak adatait. A példányokat egy *ArrayList* –be kell elhelyezni, majd ezt az *ArrayList* –et kell megadni a *DataGridView* forrásának, a következő képen:

```
dataGridView2.DataSource = felt;
```

Ahol a `felt` egy *ArrayList*, melynek elemei a feltételek tárolására készített osztály példányai. A táblázat bővítése, vagy a táblázatból való törlés nem jelent mást, mint az *ArrayList* bővítését, vagy az *ArrayList* egy elemének törlését.

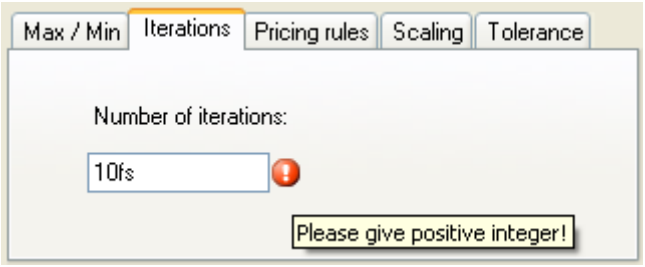
Egy másik igen hasznos komponens, amit én is használtam az *ErrorProvider,* aminek segítségével tetszőleges helyen elvégezhető a hiba ellenőrzés. Első lépésként a grafikus felületen a Form –ra kell helyezni az *ErrorProvider* –t. Ezután a kívánt esemény bekövetkezténél elvégezhetjük az ellenőrzést. Ha hibát találunk, akkor a komponens SetError függvényét kell meghívnunk. Ennek első paraméterében azt a kontrolt kell megadnunk, melyben a hibás adat van. A második paraméterben a szöveges hibaüzenetet kell átadnunk.

Például:

```
string temp = textBox1.Text;
Regex rgxHasInvalidChars = new Regex("[^0-9]");

if (rgxHasInvalidChars.IsMatch(temp))
{
    string msg = "Please give positive integer!";
    errorProvider1.SetError(textBox1, msg);
}
```

Ebben az esetben azt vizsgáljuk, hogy a `textBox1` tartalmaz-e számokon kívül más karaktert is és amennyiben igen a megadott kontrol mellett megjelenik egy kis ikon a hibát jelezve. Ha a felhasználó egérrel az ikon fölé áll, akkor egy kis súgó szövegben jelenik meg a második paraméterben megadott hibaüzenet. Az eredmény a következő lesz:



2.6 A megoldást tartalmazó fájl felépítése

A dll meghívását követően az eredmény egy fájlba íródik ki. A fájl abban a könyvtárban található, ahonnan az Lp Solver indítása történt. A fájl neve mindig az adott probléma nevével egyezik meg, a kiterjesztése pedig .sol (mint solution). A fájl valójában egy txt kiterjesztésű fájl, amelynek a szerkezete négy részre bontható.

Az első részben öt fontos dolog található, melynek egy része a feladathoz, más része a feladat megoldásához kacsolódik. Az itt található adatok a következők:

- Feladat neve
- Az optimális megoldás
- Az iterációk száma
- Az eredeti feladat sorainak a száma

- Az eredeti feladat oszlopainak a száma

A második részben egy táblázat található, amelyben a bázisban szereplő változók, valamint azok tulajdonságai szerepelnek. A változók tulajdonságai a következők:

- A változó neve
- A változó indexe
- Típusa, amely arra vonatkozik, hogy milyen megkötések voltak az adott változóra. A lehetséges értékek: FREE, LBOUNDED, UBOUNDED, BOTHBOUNDS, FIXED, amelyek rendre a következőket jelentik: szabad változó (nincs megszorítás rá), csak alsó megszorítás van, csak felső megszorítás van, alsó és felső megszorítás egyaránt, a változó értéke fix
- Intervallum, amely azt mondja meg, hogy az adott változó értéke hol helyezkedik el megkötésekben szereplő intervallumhoz képest. A lehetséges értékek: BELOW, ATLOWER, BETWEEN, ATUPPER, ABOVE, ATBOTH, amelyek rendre a következőket jelentik: alsókorlát alatt (ekkor nincs megoldás), pont alsókorláton, alsókorlát és felsőkorlát között, pont felsőkorláton, felsőkorlát felett, alsó és felső korlátot is felveszi (fix változó esetén)
- A változó értéke
- Duális változó

A harmadik részben, ugyan úgy, mint az előzőben egy táblázat található, azzal a különbséggel, hogy ebben a táblázatban a nem-bázisban szereplő változók vannak felsorolva. A változok tulajdonságai a következők:

- A változó neve
- A változó indexe
- Típusa (ugyan azokat az értékeket veheti fel, mint az előző táblázatban)
- Intervallum (ugyan azokat az értékeket veheti fel, mint az előző táblázatban)
- A változó értéke
- Redukált költség

A negyedik részben pedig az optimális érték kiszámításához szükséges idő találhat. Az idő másodpercben értendő.

3. A Szimplex módszer

3.1 A szimplex módszer ismertetése

Ha egy Lp feladatnak van optimális megoldása, akkor van optimális bázismegoldása is. Ezt a tényt használja fel az Lp feladatok numerikus megoldását megvalósító szimplex módszer. A módszer a feladatmegoldás folyamán csak bázis megoldásokat vizsgál. Amennyiben sikerül azt biztosítani, hogy egy már vizsgált bázismegoldás ne ismétlődhessen a feladat megoldása folyamán, akkor az eljárás véges számú lépést követően befejeződik. Az alap szimplex módszer csak azt biztosítja, hogy a létrehozott bázismegoldásokhoz tartozó célfüggvény értékek monoton növekvők legyenek, nem pedig azt, hogy szigorúan monoton növekvők, így nincs kizárva annak a lehetősége, hogy végtelen ciklusba fusson az algoritmus. (Megjegyzés: az alapeljárásba beépíthetőek, olyan módszerek amelyek megakadályozzák a végtelen ciklus lehetőségét, ilyen például a lexikografikus szimplex módszer.)
A szimplex módszer kanonikus alakú LP feladatok megoldására szolgál. Ez nem jelent különösebb megszorítást, hiszen minden LP feladat kanonikus alakra hozható. Először tekintsünk egy normál feladatot, mivel ebben az esetben a legkönnyebb előállítani a lehetséges kiinduló bázist (később visszatérünk a többi lehetséges esetre is):

$$\mathbf{Ax} \leq \mathbf{b}$$

$$\mathbf{x}, \mathbf{b}, \mathbf{z} \geq \mathbf{0}$$

$$\mathbf{z} = \mathbf{c}^*\mathbf{x} \rightarrow \mathbf{max}$$

A $\mathbf{z = c^*x}$ feltételt célfüggvénysornak nevezik, a z változó aktuális értéke nem más, mint az aktuális $\mathbf{x=(x_1 , x_2 \ldots x_n)}$ megoldáshoz tartozó célfüggvény érték.
A módszer alkalmazhatósága érdekében a feladatot előbb "eltérésváltozók" bevezetésével kanonikus alakra hozzuk. Minden feltételi egyenlőtlenség bal oldalához egy-egy nem negatív eltérésváltozót hozzáadva egyenlőségi feltételeket kapunk, így a feladatunk a következő alakú lesz:

$$\mathbf{[A\ E]\ [x\ u]^T = [b]^T}$$

Az egyenletrendszer indulótábláját a következő alakban írhatjuk fel:

	x_1	x_2	…	x_n	u_1	u_2	…	u_m	z	B
	a_{11}	a_{12}		a_{1n}	1	0		0	0	b_1
	a_{21}	a_{22}		a_{2n}	0	1		0	0	b_2
	…									…
	a_{m1}	a_{m2}		a_{mn}	0	0		1	0	b_m
	$-c_1$	$-c_2$	…	$-c_n$	0	0	…	0	1	0

(1.0)

Mivel a **z**-hez és az **u** vektor komponenseihez tarozó oszlopok bázis vektorok, ezért egyszerűbb alakban is felírhatjuk az (1.0) táblázatot.

	x_1	x_2	…	x_j	…	x_n	b
u_1	a_{11}	a_{12}		a_{1j}		a_{1n}	b_1
u_2	a_{21}	a_{22}		a_{2j}		a_{2n}	b_2
…							…
u_i	a_{i1}	a_{i2}		a_{ij}		a_{in}	b_i
…							…
u_m	a_{m1}	a_{m2}		a_{mj}		a_{mn}	b_m
z	$-c_1$	$-c_2$	…	$-c_j$	…	$-c_n$	0

(2.0)

Erről a táblázatról az **x = 0 , u = b , z = 0** megoldás olvasható le. A szimplex tábla szerkezete az első és minden közbeeső iterációban a (2.0) táblázat szerkezetével fog megegyezni. Az elő oszlopban, a bázisban szereplő változók nevei találhatóak. Az első sorban pedig nem a bázisban szereplő változók találhatóak. A táblázat j. oszlopának elemi az $[A^j \ -cj]^T$ vektor koordinátái, ebben a bázisban.

A kiinduló szimplex tábla birtokában a következő iterációs lépeseket kell ismételni:

1. **lépés:** *Kiválasztjuk a bázisba bekerülő oszlopvektort.* Keresünk a táblázat alsó sorában szereplő elemek között negatív elemet. Ha nincs ilyen elem, akkor optimális megoldást találtunk, leolvasható a táblázatból a megoldás és az eljárás végetér. Ha van ilyen elem, akkor kiválasztunk közülük egyet: $cj<0$ és eldöntjük, hogy az A^j oszlopvektort vonjuk be bázisba. (Ez a feltétel biztosítja azt, hogy a célfüggvény érteke monoton növekvő legyen)

2. **lépés:** *Kiválasztjuk a bázisból kikerülő vektort.* Keresünk az A^j oszlopvektor elemei között, olyan elemet, amelyre teljesül az alábbi két feltétel:

   - (α) $a_{ij} > 0$, azaz a generáló elem csak pozitív szám lehet (Ha nincs ilyen, akkor megállapítjuk, hogy a célfüggvény nem korlátos, így nincs optimális megoldás és az eljárás végetér)

   - (β) b_i / a_{ij} minimális legyen, azaz ha az adott (j-edik) oszlop pozitív elemeivel osztjuk a **b** oszlop megfelelő elemeit, akkor generáló elemnek az oszlop azon elemét kell választani, amelyre ez a hányados a legkisebb. (Ha több ilyen is van, akkor közülük tetszőlegesen választunk egyet.)

   Ha az a_{ij} elemre teljesül mind az (α) és a (β) feltétel, akkor az x_j változóhoz tarozó oszlopvektor az u_i változóhoz tarozó sorvektor helyére fog kerülni a bázisban, amelyet a bázistranzformáció segítségével érünk el.

3. **lépés:** *Végrehajtjuk a bázis cserét.* A tábla új elemeit a következő módon kapjuk, ha a generáló elemnek az a_{ij} elemet választottuk:

   Az u_i helyére beírjuk x_j–t és x_j helyére beírjuk u_i–t.

   $a_{ij(új)} = 1 / a_{ij(régi)}$

   $a_{xy(új)} = a_{xy(régi)} - (a_{xj(régi)} * a_{iy(régi)}) / a_{ij(régi)}$; $x \neq i$, $y \neq j$

   $a_{iy(új)} = a_{iy(régi)} / a_{ij(régi)}$

   $a_{xj(új)} = - a_{xj(régi)} / a_{ij(régi)}$

   Példa:

Tegyük fel, hogy a normál feladathoz tartozó

	x_1	…	x_r	x_{r+1}	…	x_n	b
u_1	a_{11}		a_{1r}	$a_{1,r+1}$		a_{1n}	b_1
…							…
u_r	a_{r1}		a_{rr}	$a_{r,r+1}$		a_{rn}	b_r
u_{r+1}	$a_{r+1,1}$		$a_{r+1,r}$	$a_{r+1,r+1}$		$a_{r+1,n}$	b_{r+1}
…							…
u_m	a_{m1}		a_{mr}	$a_{m,r+1}$		a_{mn}	b_m
z	$-c_1$	…	$-c_r$	$-c_{r+1}$	…	$-c_n$	0

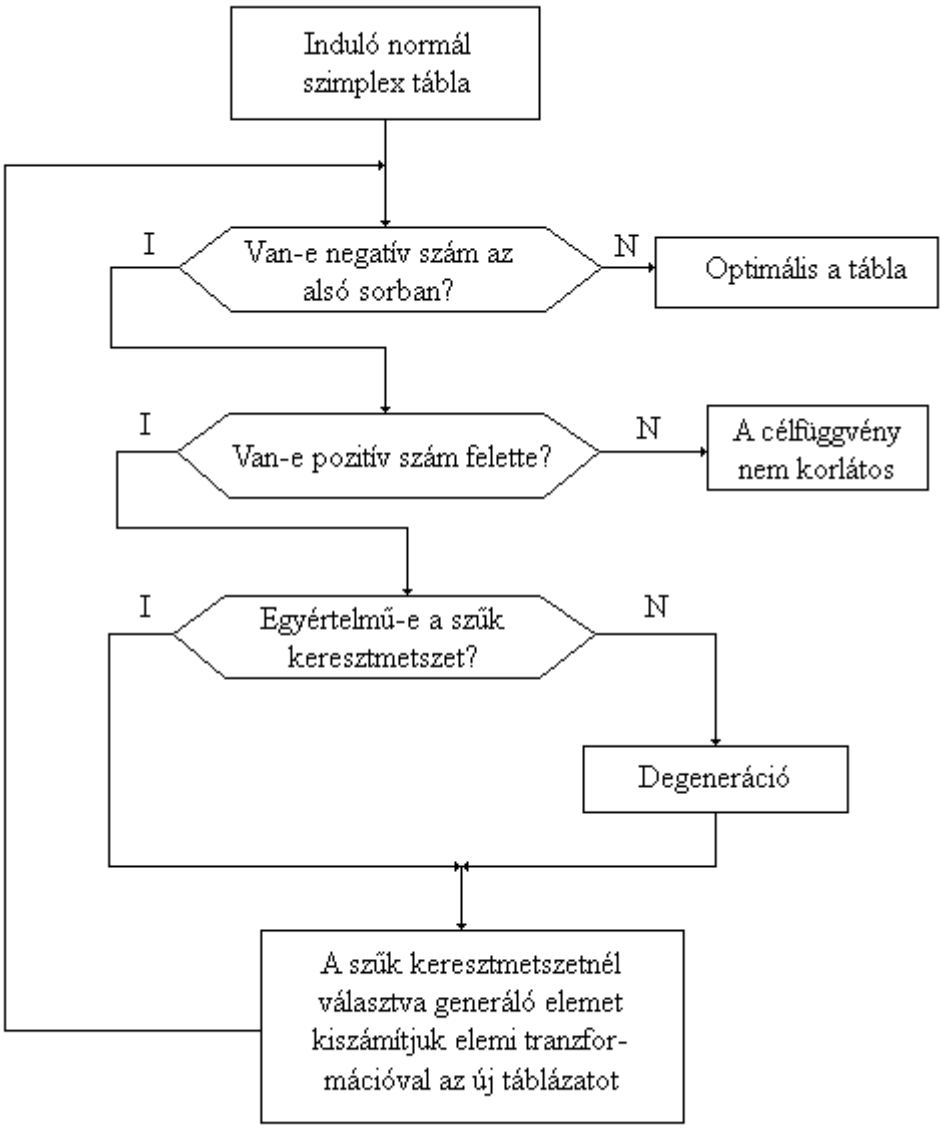
táblázatból kiindulva r db elemi bázistarnzformációt hajtottunk végre, generáló elemeket rendre az első r sorban, illetve oszlopban választva (ez sor- és oszlopcserével mindig elérhető). Ekkor a fenti táblázat a következő tömörebb alakban írható fel:

	x_1	x_2	b
u_1	A_{11}	A_{12}	b_1
u_2	A_{21}	A_{22}	b_2
z	c_1	c_2	0

A bázistarnzformációt végrehajtva a táblázat a következő képen alakul:

	u_1	x_2	B
x_1	$1 / A_{11}$	A_{12} / A_{11}	b_1 / A_{11}
u_2	$- A_{21} / A_{11}$	$A_{22} - A_{21} * A_{12} / A_{11}$	$b_2 - A_{21} * b_1 / A_{11}$
z	$- c_1 / A_{11}$	$c_2 - c_1 * A_{12} / A_{11}$	$- c_1 * b_1 / A_{11}$

3.2 A szimplex módszer blokksémája



3.3 Példa szimplex módszerre

Tekintsük a következő normál feladatot:

$$15\,x_1 + 20\,x_2 \leq 1440$$
$$3\,x_1 + 2\,x_2 \leq 240$$
$$x_2 \leq 50$$
$$600\,x_1 + 500\,x_2 \rightarrow \max \qquad (x \geq \mathbf{0})$$

Ennek a feladatnak az indulótábláját a következő formában írhatjuk fel:

	x_1	x_2	b
u_1	<u>15</u>	20	1400
u_2	3	2	240
u_3	0	1	50
z	-600	-500	0

Az induló táblán végrehajtjuk a szimplex módszer első lépést, azaz megkeressük a bázisba kerülő oszlopvektort. Mivel az alsó sor mindkét eleme negatív, ezért bármelyiket választhatnánk, de a Dantzing féle klasszikus oszlopválasztási szabálynak megfelelően válasszuk az x_1 oszlopot, ahol a legmagasabb a c_i abszolút értéke. Ezután hajtsuk végre a szimplex módszer második lépést és döntsük el az oszlop melyik eleme legyen a generáló elem. Az (α) feltételnek csak a 15 és a 3 tesz eleget. A (β) feltételnek pedig a 3, mivel b_i / a_{ij} ebben az esetben minimális (240/3=80), tehát a generáló elem a 3 lesz. Ezzel eldőlt, hogy melyik vektor fog bázisba kerülni és melyik vektor fog kikerülni onnan. Hajtsuk végre a szimplex módszer harmadik lépését. A báziscserét követően a következő szimplex táblát kapjuk:

	u_2	x_2	b
u_1	-5	<u>10</u>	240
x_1	1/3	2/3	80
u_3	0	1	50
z	200	-100	48000

Ezután az első lépessel kezdjük előröl a szimplex módszert. Az abban szereplő feltételnek csak a második oszlop felel meg, mivel egyedül az tartalmaz negatív számot az utolsó sorban. Ezt követően a szimplex módszer második lépésében eldöntjük a generáló elemet. Az (α) és

(β) feltételnek csak a 10 tesz eleget, így ugorhatunk a harmadik lépésére. A báziscserét követően a következő szimplex táblát kapjuk:

	u_2	u_1	b
x_2	-1/2	1/10	24
x_1	2/3	-1/15	64
u_3	1/2	-1/10	26
z	150	10	50400

Ekkor megint az első lépessel kell folytatnunk az eljárást. Mivel az utolsó sorban csupa pozitív szám áll, ezért optimális megoldáshoz jutottunk. A feladatunk megoldása a következő:

$$x_1 = 64 \; ; \; x_2 = 24;$$
$$Z = 50400$$
$$u = [0,0,26]$$

3.4 Kiinduló lehetséges bázis előállítása

A szimplex módszer ismertetésénél normál feladatból indultunk ki, ahol kulcsszerepe volt annak, hogy már az indulótábla lehetséges bázismegoldást ad, tehát az **$Ax \leq b \; ; \; c^*x \rightarrow max \; ; \; b \; , \; x \geq 0$** normál feladathoz tartozó kanonikus alak:

$$[A \; E] \; [x \; u]^T = [b]^T \quad ; \; (c,0)^*[x \; u]^T \rightarrow max$$

már automatikusan tartalmaz egy kiinduló lehetséges bázist, amelyet az $u_1, \ldots, u_m$ változókhoz tarozó egységvektorok alkotnak. A lehetséges megoldás az $x_j = 0$, $j = 1...n$; $u_i = b_i$, $i = 1...m$.
Azonban ha a megoldandó feladat módosított normál, vagy általános alakú, akkor a kiinduló lehetséges bázis előállítása külön megfontolást és eljárást igényel. Az első fázisban arra törekszünk, hogy egy kiinduló lehetséges bázismegoldást létrehozzunk, majd a második fázisban megkeressük az optimális megoldást. Ezt az eljárást szokták *kétfázisú szimplex módszer*nek nevezni.

1) Először foglalkozzunk a **módosított normál feladattal**:

$$\mathbf{A_1x = b_1}$$
$$\mathbf{A_2x \le b_2} \qquad \mathbf{x \ge 0,} \qquad \mathbf{b_1 \ge 0,} \qquad \mathbf{b_2 \ge 0}$$
$$\mathbf{c^*x \rightarrow max}$$

A feltételi egyenlet-, illetve egyenlőtlenség rendszert egyenletrendszerré alakíthatjuk (az egységes tárgyalásmód érdekében az egyenleteknél is alkalmazzuk az eltérésvektort):

$$\mathbf{A_1x + \breve{u}_1 = b_1}$$
$$\mathbf{A_2x + u_2 \le b_2} \qquad \mathbf{x \ge 0,} \qquad \mathbf{b_1, b_2 \ge 0,} \qquad \mathbf{\breve{u}_1, u_2 \ge 0}$$
$$\mathbf{c^*x \rightarrow max} \qquad (3.0)$$

Az $\breve{u}_1$ vektornál „kalappal” jeleztem, hogy az egyenletrendszerhez tartozik, így komponensei csak zérusok lehetnek. Ennek alapján az indulótábla:

	x	
$\breve{u}_1$	A_1	b_1
u_2	A_2	b_2
z	-c	0

Ebből a táblából kapható bázismegoldások azonban a

$$\mathbf{A_1x \le b_1}$$
$$\mathbf{A_2x \le b_2} \qquad \mathbf{x \ge 0,} \qquad \mathbf{b_1 \ge 0,} \qquad \mathbf{b_2 \ge 0}$$

normál feladat bázismegoldásai, ezért valamilyen módon érvényt kellene szerezni az $\mathbf{\breve{u}_1=0}$ feltételnek. Nézzük a következő normál feladatot:

$$\mathbf{A_1x \le b_1}$$
$$\mathbf{A_2x \le b_2} \qquad \mathbf{x \ge 0,}$$
$$\mathbf{\hat{g}(x) = -1^*A_1^*x \rightarrow max} \qquad (4.0)$$

A feltétel rendszer alapján:

$$\hat{g}(x) = -1*A_1*x \leq -1*b_1$$

másrészt **x** akkor lehet csak megoldása módosított normál feladatnak, ha

$$-1*A_1*x = -1*b_1$$

fennáll, és ekkor $\check{u}_1 = 0$. Ezzel megfogalmazhatjuk a következő állítást: A (3.0) egyenletrendszernek – és az eredeti módosított normál feladatnak – akkor és csak akkor van lehetséges megoldása, ha a (4.0) normál feladat maximális értéke $-1*b_1$.
Mivel a (3.0) és a (4.0) feladatok indulótáblái csak az alsó sorban (célfüggvényben) különböznek egymástól, ezért egyetlen táblázatot elég készíteni, szerepeltetve rajta mind a két célfüggvényt.:

	X	
u_1	A_1	b_1
u_2	A_2	b_2
z	-c	0
ž	$-1*A_1$	0

Az alsó sorban lévő célfüggvényt **másodlagos célfüggvénynek** nevezzük. Első feladatunk eljutni egy lehetséges megoldáshoz. A fentiek miatt először a másodlagos célfüggvény optimumának keresését tartjuk szem előtt. A $\hat{g}(x)$ korlátos, tehát mindig van optimális megoldása, csak az lehet a probléma, hogy ez kisebb $-1*b_1$ –nél. Ilyenkor a lehetséges halmazok megoldása üres és nem kell végrehajtani a második fázist.

2) Térjünk át az **általános feladatra**:

$$A_1x = b_1$$
$$A_2x \leq b_2$$
$$A_3x \geq b_3 \qquad x \geq 0, \qquad b_1 \geq 0, \qquad b_2 \geq 0 \qquad b_3 \geq 0$$
$$c*x \rightarrow max$$

Vezessünk be egy $\mathbf{v} \geq \mathbf{b}$ vektort, amelyre teljesül, hogy $\mathbf{A_3x - v \geq b_3}$
Ekkor már az

$$\mathbf{A_1x = b_1}$$
$$\mathbf{A_2x \leq b_2} \qquad \mathbf{x \geq 0,} \qquad \mathbf{v \geq 0,}$$
$$\mathbf{[A_3\ -E]\ [x\ v]^T = b_3}$$
$$\mathbf{c^*x \rightarrow max}$$

Módosított normál feladatról van szó, amelynek induló táblája:

	x	v	
ŭ$_1$	A_1	0	b_1
u$_2$	A_2	0	b_2
ŭ$_3$	A_3	-E	b_3
z	-c	0	0
ž	$-1^*A_1 - 1^*A_3$	1	$-1^*b_1 - 1^*b_3$

Ahol az 1) pontban ismertetett eljárást követjük, azaz először a másodlagos célfüggvény optimumának keressük, majd az elsődlegesét.

4. MPS formátum

Ez az egyik legelterjedtebb fájl formátum, amelyet LP feladatok megadásához használnak. Az általam használt „*retrolP”* motor is ilyen formátummal dolgozik. Ennek az a nagy előnye, hogy számos feladatot lehet megtalálni ebben a formátumban, amely nagyban megkönnyíti a tesztelést.

Maga a formátum nagyon hosszú múltra tekint vissza, ugyanis anno ezt még a lyukkártyás rendszereknél kezdték el használni. Az MPS formátumot (Mathematical Programming System) eredetileg az IBM vezette be, hogy általánosítsa a lineáris programozás és az egész értékű programozás felírását. A legfontosabb dolog, amit tudni kell az MPS formátumról, az hogy ez egy oszlop orientált formátum, és a hagyományostól teljesen eltérő megközelítés miatt nehezen átlátható. Ami elsőre szembetűnő különbség az MPS és a hagyományos formátum között, azaz hogy az MPS formátumban minden külön nevet kap, így nem csak a változók rendelkeznek saját évvel, hanem a megszorítások is. Ezek a nevek nem elsősorban a programnak szólnak, így célszerű beszédes neveket megadni, hogy később könnyebben áttekinthető legyen mind a feladat, mind a megoldás. A formátum lyukkártyás múltja miatt szigorú szabályok vannak az adatok elhelyezkedésére. Minden egyes sorban maximum 6 mező lehetséges, amelyek a következő intervallumokba esnek: 2-3 ; 5-12 ; 15-22 ; 25-36 ; 40-47 ; 50-61. A mezők megadásánál eredetileg csupa nagybetűt kellett használni, azonban mára már számos program kezelni tudja mind a kis- és nagybetűket. Azonban célszerű továbbra is csupa nagybetűt használni, ugyanis sosem lehet tudni, mikor kell olyan programmal megoldani a feladatot, amely csak a nagybetűket fogadja el. A fájl sorokból épül fel, amely sorokat különböz szegmensekre lehet osztani. A szegmensek szigorú sorrendben követik egymást (a lyukkártyás múlt miatt). Minden egyes szegmens kezdetét egy olyan sor jelzi, amely csak a szegmens nevét tartalmazza és egészen a következő szegmens nevéig tart. Hét különböző szegmens van, amelyek rendre a követezőek (sorrendjük kötött):

- NAME: Ez a szegmens csak egy sort tartalmaz. A sor elején található a szegmens megnevezése, a harmadik mezőben található az LP feladat neve.

- ROWS: Ebben a szegmensben adhatóak meg a megszorítások nevei, valamint a célfüggvény neve. Minden egyes megszorításhoz a nevén kívül tarozik egy típus is. A

megszorítás típusa az első mezőbe kerül, a megszorítás neve pedig a második mezőbe. A típusok a következők lehetnek:

Típus	Jelentés
E	Egyenlő
L	Kisebb egyenlő
G	Nagyobb egyenlő
N	Célfüggvény
N	Nincs megszorítás

Annyit érdemes megjegyezni, hogy csak az első **N** típusú sor lesz célfüggvény, az összes többi N típusú sor nem tartalmaz majd megszorítást.

- COLUMNS: Ebben a szegmensben történik meg a nem nulla változók, valamint a célfüggvény együtthatóinak a definiálása. A nulla együtthatójú változok megadása szükségtelen, de nem tilos. A sorok a következő struktúrában épülnek fel.: A második mező tartalmazza, hogy mely megszorításban szerepel a változó, a harmadik mező tartalmazza a változó nevét, a negyedik mező tartalmazza az adott megszorításban az adott változó együtthatóját. Az ötödik és hatodik mező opcionális. Amennyiben van, akkor az ötödik mező egy újabb változó nevet tartalmaz, a hatodik mező pedig a hozzá tartozó együtthatót.

- RHS: Ebben a szegmensben lehet megadni a megszorítások jobb oldalát (a **b** vektor elemeit). Elsősorban a **b** vektor nem nulla elemeinek a megadása történik itt. Amennyiben nem adunk értéket egy a ROWS szegmensben szereplő megszorítás jobb oldalának, akkor az automatikusan nulla lesz. Az RHS sorok szerkezete hasonló, mint a COLUMNS –é, azzal a kivétellel, hogy itt a második mezőben az RHS neve található és a harmadik mezőben található a megszorítás neve, amihez tartozik a negyedik mezőben szereplő érték. (Itt is opcionális az ötödik és hatodik mező.) Van arra is lehetőségünk, hogy több **b** vektort is megadjunk, azonban ebben az estben jelezni kell, hogy melyik legyen az alapértelmezett.

- RANGES: Ennek a szegmensnek a segítségével lehet megszorításokhoz egyaránt alsó és felső korlátot megadni. Ha a megszorításunkra igaz hogy: $h \leq a_{i1}*x_1 + \ldots + a_{in}*x_n \leq u$, akkor azt mondjuk, hogy a megszorításunk hatásköre $r = u - h$. Ezt a hatáskört lehet a RANGES szegmensben tárolni. Az r önmagában nem elegendő az alsó és felső korlát meghatározáshoz, mivel még ismerni kell vagy u-t, vagy h-t, de ez nem jelenet problémát mivel már az RHS szegmensben egyikük tárolásra került. Az egyes sorok struktúrája megegyezik a COLUMNS –éval, azzal a kivétellel, hogy a második mezőbe a RANGES neve kerül. Ha b az RHS szegmensben lett megadva és r a RANGES szegmensben, akkor u és h a következő képen alakul:

Row Típusa	R előjele	h	H
G	+ / -	b	b + | r |
L	+ / -	b - | r |	B
E	+	b	b + | r |
E	-	b - | r |	B

Ezt a szegmenst nem kötelező szerepeltetni a fájlba, mivel opcionális.

- BOUNDS: Ebben a szegmensben tudjuk megadni a változók alsó és/vagy felső korlátjait. Amennyiben egy változót nem szerepeltetünk a BOUNDS szegmensben, akkor a változó nulla és végtelen közé esik. A BOUNDS szegmens sorainak struktúrája a következő.: Az első mező a BOUNDS típusa, második mező a BOUNDS neve, a harmadik mező a változó neve, a negyedik mező a BOUNDS értéke. A BOUNDS típusai a következők lehetnek:

Típus	Jelentés
LO	$b \leq x$
UP	$x \leq b$
FX	$x = b$
FR	szabad változó
MI	$\infty^- < x$
BV	x = 0 vagy x = 1

Ezt a szegmenst sem kötelező szerepeltetni a fájlban, mivel ez is opcionális.

- ENDATA: Ez szegmens jelzi, hogy véget ért a fájl.

A könnyebb érthetőség kedvért egy egyszerűbb példán keresztül is bemutatom az MPS és a hagyományos alak közti különbséget.

4.1 Példa MPS formátumra

```
NAME          TESTPROB
ROWS
 N  COST
 L  LIM1
 G  LIM2
 E  MYEQN
COLUMNS
    XONE      COST                 1   LIM1                 1
    XONE      LIM2                 1
    YTWO      COST                 4   LIM1                 1
    YTWO      MYEQN               -1
    ZTHREE    COST                 9   LIM2                 1
    ZTHREE    MYEQN                1
RHS
    RHS1      LIM1                 5   LIM2                10
    RHS1      MYEQN                7
BOUNDS
 UP BND1      XONE                 4
 LO BND1      YTWO                -1
 UP BND1      YTWO                 1
ENDATA
```

Hagyományos alakban a példa:

```
Célfüggvény
 COST:    XONE + 4 YTWO + 9 ZTHREE
Megszorítások
 LIM1:    XONE + YTWO <= 5
 LIM2:    XONE + ZTHREE >= 10
 MYEQN:   - YTWO + ZTHREE  = 7
Változók megszorításai
 0 <= XONE <= 4
-1 <= YTWO <= 1
```

5. Alternatív oszlop választási szabályok

A bázisba kerülő oszlopvektor kiválasztásakor bármelyik nem a bázisban szereplő oszlopvektort kiválaszthatnánk. Azonban a kiválasztás módjára több módszer is létezik A különböző módszerek az optimális értéket más-más úton érik el, ennek köszönhetően a szimplex módszer iterációinak a száma is más és más. Három különböző megközelítési módot tekintünk, hogyan lehet kiválasztani a bázisba bekerülő elemet.

5.1 Klasszikus oszlop választási módszer (Dantzing column choice) :

A Dantzig által használt eredeti szabály szerint azon oszlop került bázisba, amelyikben a c_{ij} érték abszolút értéke a legnagyobb. A nem bázisban lévő változó kiválasztásának ez a módja adja a legnagyobb növekedést a célfüggvény / θ értékében (Ahol $\theta = \min b_i / a_{ij}$). Ez a kritérium nagyon egyszerű és nagyon könnyen kiszámítható. Más eljárásokkal szemben, anélkül választunk oszlopot, hogy figyelembe vennénk az aktuális együtthatókat, azaz az a_{ij} – ket.

5.2 A legmeredekebb él módszere (Steepest Edge):

Minden egyes lehetséges oszlopvektorhoz kiszámítjuk a következő hányadost. Az oszlopok közül pedig azt választjuk, ahol ez a hányados a legkisebb.

$$c_j / \sqrt{1+\sum_i a_{ij}^2} .$$

(A gyakorlatban a gyökvonást el szokták hagyni, mivel kerekítési hibákhoz vezethet és a futási időt is növeli. Azzal hogy elhagyjuk nem követünk el hibát, mivel úgyis csak az összehasonlítások miatt számoljuk ki). A legmeredekebb él módszere több műveletet és időt igényel, mind a klasszikus, mind a módosított szimplex módszer esetén. A módosított szimplex módszernél az a_{ij} elemek nehezen elérhetőek. Így akár úgy is tűnhetne, hogy ez a módszer nem használható módosított szimplex módszer esetében, azonban egy ügyes rekurzív eljárásnak köszönhetően implementálható a módszer. [Forest and Goldfarb, 1992]. (A Harris féle *Devex* szabály megközelíti a legmeredekebb él módszerének hatékonyságát.) Minden esetre a normál szimplex módszer esetén az együtthatók könnyen elérhetőek. A módszer

hatékonyságával kapcsolatban egy dolgot meg kell jegyeznünk. Elegendő a fenti hányadost csak a lehetséges oszlop vektorokhoz kiszámítani. A nem lehetséges oszlopok kizárása jelentős megtakarítást eredményez. Amikor a legmeredekebb él módszerét alkalmazzuk fontos, hogy legyen valamilyen információnk, vagy sejtésünk arról, hogy hány lehetséges, nem bázisban lévő vektorunk van.

5.3 A legnagyobb változás módszere:

Minden egyes lehetséges oszlopnál végre kell hajtani a szimplex módszer második lépését, azaz el kell dönteni, hogy melyik vektor kerülne ki a bázisból. Minden egyes oszlopban, ezek után, már csak egy elemet kell vizsgálni. Az egyes elemekhez ki kell számolni, hogy mennyivel változna a célfüggvény értéke, ha az adott elem lenne a generáló elem. Ez így kapott értékek közül a legnagyobbat választva fogunk legközelebb jutni az optimális értékhez. Ezért szokták ezt a legnagyobb változás módszerének hívni. A módszer számítás igénye még a *legmeredekebb él* módszerénél is nagyobb. Az eredmények azt mutatják, hogy a hatásfoka nem jobb a *legmeredekebb él* módszerénél, így ezt az eljárást kevésbé használják. Mindamellett hogy könnyen implementálható, a módosított szimplex módszerben sem elterjedt.

6. Skálázás

Ebben a részben két módszert kerül tárgyalásra, amelyek segítségével a skálázási faktorokat (p) kiszámolhatjuk. Mindkét technika számos kereskedelmi és ingyenes - LP feladat megoldását megvalósító - programban megtalálható. Azonban mielőtt rátérnénk a lényegre, ejtsünk arról pár szót mi is az a skálázás.

Mivel a számítógép véges pontossággal tudja kezelni a számokat, ezért kerekítési hibák léphetnek fel a numerikus számítások alatt. Ezek a kerekítési hibák elsőre nem tűnnek jelentősnek, azonban ha további számításokat végzünk velük, döntően befolyásolhatják a kapott eredményt, amely a valós életben akár végzetes hibát is okozhat. A szimplex módszernél ennek a veszélye nem elhanyagolható, sőt bizonyos esetekben nagyon is magas, mivel a módszer sajátja a nagyszámú iteráció. Így a szimplex módszer elején egy aprónak tűnő kerekítési hiba a módszer végére hatalmas méretűvé nőhet. A skálázási eljárás célja,

hogy ezen kerekítési hibákat megelőzze. A eljárás alapötlete az, hogy az induló táblát úgy kell átalakítani, hogy a kerekítési hibákat elkerüljük és az átalakított induló táblából kapott eredményt vissza lehessen alakítani úgy, hogy az eredeti LP feladat megoldása legyen. Mivel az induló tábla átalakításának is van erőforrás igénye, ezért nem minden esetben célszerű végrehajtani azt. Annak érdekében, hogy el tudjuk dönteni, mikor használjuk a skálázást és mikor ne, bevezethető minden mátrixhoz egy mennyiség, amely jelzi, hogy milyen mértékű pontatlanság várható. Ezt az értéket a következő képen lehet definiálni:

$$\sigma(A) = \frac{\max_{i,j \in J_+}\left(\left|a_{ij}\right|\right)}{\min_{i,j \in J_+}\left(\left|a_{ij}\right|\right)}$$

ahol $J_+ = \{i , j \mid a_{ij} \neq 0\}$. Az érték, tehát nem más, mint a mátrixban a legnagyobb és legkisebb szám abszolút értékeinek hányadosa. Minél nagyobb ez a hányados, annál rosszabbul skálázott a mátrix, annál nagyobb valószínűséggel várható pontatlanság a feladat megoldása folyamán. Ezt összefoglalva a következő definíció mondható el:

Definíció: Egy mátrixról akkor mondjuk, hogy rosszul skálázott, ha $\sigma(A) \geq 10^5$

A skálázás célja az, hogy σ(A) értékét olyan kicsire csökkentse, amennyire csak lehet. Ahhoz, hogy ezt elérjük, skáláznunk kell a mátrixot annyiszor, amíg a σ(A) értéke 10^5 alá nem csökken.. A skálázás folyamata következő képen történik:

1. Kiszámoljuk a sorokhoz tartozó skálázási faktort $\rho=(\rho_1 , \rho_2 \ldots \rho_m)$
2. A mátrix i -edik sorának összes elemét elosztjuk ρ_i – vel. (i= 1,2 … m), Valamint $\rho\_row_i = \rho\_row_i * \rho_i$ (i= 1,2 … m)
3. Ellenőrizzük, hogy *σ(A)* értéke elég kicsi, ha igen, akkor végeztünk.
4. Kiszámoljuk az oszlopokhoz tartozó skálázási faktort $\rho'=(\rho'_1 , \rho'_1 \ldots \rho'_n)$
5. A mátrix j -edik oszlopának összes elemét elosztjuk ρ'_j – vel. (j= 1,2 … n), Valamint $\rho\_col_j = \rho\_col_j * \rho_j$ (j= 1,2 … n)
6. Ellenőrizzük, hogy *σ(A)* értéke elég kicsi, ha igen, akkor végeztünk, ha nem, akkor kezdjük előröl a folyamatot az első lépéssel.

A ρ_row és a ρ_col vektor szolgál arra, hogy a skálázott mátrix által kapott eredményből vissza tudjuk majd nyerni az eredeti feladat eredményét.
A skálázási vektorok előállítására számos módszer létezik, ezek közül két hatékony módszert fogok most részletesen ismertetni:

I. Geometriai módszer:

A módszer szerint a következő ρ^r oszlop-vektort definiáljuk a sorok skálázási vektortoraként:

$$\rho^r = (\rho^r_1, \rho^r_2, \dots, \rho^r_j)^T$$

Ahol

$$\rho_i^r = \left(\prod_{j \in J_i^+} a_{ij} \right)^{1/K_i^r} \qquad i = 1,2,...,m;$$

$J^+_j = \{j : a_{ij} \neq 0\}$, i = 1 , 2 , … , m, egy sorokhoz kacsolódó halmaz, amelynek azon a_{ij} elemek j. indexe lesz az eleme, amelyikre teljesül, hogy $a_{ij} \neq 0$. K^r_i az i. sor nem nulla elemeinek a darabszáma.

Analóg módon, az oszlopok skálázáshoz a következő ρ^c sor-vektort definiáljuk:

$$\rho^c = (\rho^c_1, \rho^c_2, \dots, \rho^c_j)$$

Ahol

$$\rho_j^c = (\prod_{i \in I_j^-} a_{ij})^{1/K_j^c} \qquad j = 1,2,...,n;$$

$I^+_j = \{i : a_{ij} \neq 0\}$, j = 1 , 2 , … , n, egy oszlopokhoz kacsolódó halmaz, amelynek azon a_{ij} elemek i. indexe lesz az eleme, amelyikre teljesül, hogy $a_{ij} \neq 0$. K^r_j az j. sor nem nulla elemeinek a darabszáma.

II. Középérték módszer:

Ez egy másik alternatíva a skálázási vektor kiszámítására. Csak úgy, mint az előző esetben most is ρ^r oszlop-vektort definiáljuk a sorok skálázási vektoraként:

$$\rho^r = (\rho^r_1, \rho^r_2, \dots, \rho^r_j)^T$$

Ahol

$$\rho_i^r = \sqrt{r_i' r_i''}, \quad i = 1,2,...,m;$$

$$r_i' = \max_{j:a_{ij}\neq 0} |a_{ij}|, \quad r_i'' = \min_{j:a_{ij}\neq 0} |a_{ij}|, \quad i = 1,2,...,m.$$

Analóg módon, az oszlopok skálázáshoz a következő ρ^c sor-vektort definiáljuk:

$$\rho^c = (\rho^c{}_1, \rho^c{}_2, \ldots, \rho^c{}_j)$$

Ahol

$$\rho_j^c = \sqrt{c_j' c_j''}, \quad j = 1,2,...,n;$$

$$c_j' = \max_{i:a_{ij}\neq 0} |a_{ij}|, \quad c_j'' = \min_{i:a_{ij}\neq 0} |a_{ij}|, \quad j = 1,2,...,n.$$

7. Összefoglalás

Diplomamunkám alapvető célja az volt, hogy egy olyan grafikus környezetet, hozzak létre, amely egy már létező, az LP feladatok megoldását megvalósító programcsomagra épül. Továbbá az is cél volt, hogy egy olyan grafikai felületet biztosítsak, ahol könnyedén lehet módosítani az LP feladatban szereplő változók és megszorítások tulajdonságait. Szintén elvárás volt az is, hogy a szimplex módszert megvalósító külső program paramétereinek a beállítására is biztosítsak egy felületet. Ezeket a célokat teljes egészében sikerült elérni és egy olyan programcsomagot létrehozni, amely nagy segítség mindazok számára, akik valamilyen LP feladatot kívánnak megoldani. A szimplex módszert megvalósító külső fájl gondos megválasztásnak köszönhetően, egy igen pontos és megbízható eredményt kap az igazán nehéz feladatokra is, ráadásul igen gyorsan. A sikeres megvalósítás mellett volt egy probléma, amellyel a tesztelések alatt találkoztam. A probléma abban az esetben jelentkezett, amikor a RetroLP –t megvalósító dll –t egymásután több különböző MPS fájlra hívtam meg. Ekkor a végeredmény bázis és nem bázis vektoraiban megjelentek az előző feladat változói és megszorításai. Sajnos ez valamilyen memóriakezelési hibára vezethető vissza az eredeti RetroLP programcsomagban. Erre azért nem derült eddig fény, mert az eredeti konzolos változatban egy futtatás alkalmával csak egy feladatot oldott meg a program. A nyílt forráskódnak köszönhetően magam is utána néztem mi lehet az oka a jelenségnek, azonban a megoldást nem sikerült megtalálnom. A probléma megszüntetése érdekében felvettem a kapcsolatot a RetroLP készítését végző Richard Van Slyke professzorral, aki igen készségesen és segítőkészen ált a dolgokhoz. Jelenleg ő is és én is azon dolgozunk, hogy rájöjjünk a hiba forrására és kijavítsuk azt.

A diplomamunkám másik célja az volt, hogy egy részletes leírást adjak a RetroLP elméleti működéséről, azaz az eredeti szimplex módszerről. A leírást próbáltam, úgy megvalósítani, hogy azt bárki könnyedén megérthesse. Azt hiszem ez sikerült is.

Az MPS fájl bemutatásánál törekedtem arra, hogy minél részletesebben írjam le a fájl felépítését, ugyanis hosszas keresés ellenére sem találtam magyar fordítást a témában. Remélem a leírásom hasznos lesz mindazok számára, akiknek valaha szüksége lesz információra a fájl felépítésről.

A szimplex módszert javító eljárások közül kettőre tértem ki részletesen. Az első az alternatív oszlopválasztási szabályok, amelyet sokkal gyakrabban alkalmaznak a szimplex módszer implementációjaiban. Ennek az oka az, hogy nagymértékben gyorsítja a szimplex módszert. A második a skálázás volt, amely jelentősége kisebb, ugyan de vannak olyan feladatok, ahol a pontosság megtartásának érdekében elkerülhetetlen a használata.

Irodalom Jegyzék

Bixby, Robert E., "Implementing the Simplex Method: The Initial Basis," *ORSA J. on Computing*, *Vol. 4*, No. 3, pp. 267-284, Summer, 1992.

Chvátal, Vasek, *Linear Programming*, Freeman, 1983.

Dongarra and Francis Sullivan, "Guest Editor's Introduction: The Top Ten Algorithms," *Computing in Science and Engineering*, pp. 22-23, January/February, 2000.

Eckstein, J., I. Bodurglu, L. Polymenakos, and D. Goldfarb, "Data-Parallel Implementations of Dense Simplex Methods on the Connection Machine CM-2," *ORSA Journal on Computing*, v. 7, *n*. 4, pp. 402-416, Fall 1995.

Forrest, John and Donald Goldfarb, "Steepest-edge simplex algorithms for linear programming," *Mathematical Programming, vol. 57*, pp. 137-150, 1992.

Harris, P. M. J., "Pivot selection methods of the Devex LP code," *Mathematical Programming, 5,*pp. 1-28, 1973.

Murtagh, Bruce A., and Michael Saunders, "MINOS 5.5 Users Guide," Technical Report SOL 83-20R, Revised July, 1998.

Thomakakis, Michael E., and Jyh-Charn Liu, "An Efficient Steepest-Edge Simplex Algoirithm for SIMD Computers," International Conference on Supercomputing, Philadelphia, pp. 286-293, 1996.

Yarmish, Gavriel, *A Distributed Implementation of the Simplex Method*, Ph.D. dissertation, Polytechnic University, Brooklyn, NY, March, 2001.

Bajalinov,E.B., "Linear-Fractional Programming: Theory, Methods, Applications and Software", Kluwer Academic Publishers, 2003.

Curtis,A.R., Reid,J.K., "On the Automatic Scaling of Matrices for Gaussian Elimination", J. Inst. Maths. Applics., Vol.10, pp.118-124, 1972.

Bradley, P.S., Usama M. Fayyad, and O.L. Mangasarian, "Mathematical Programming for DataMining: Formulations and Challenges," *INFORMS Journal on Computing*, Volume: 11. Summer1999, Number: 3. pp. 0217-238, 1999.

Skeel,R.D., "Scaling for Stability in Gaussian Elimination", J. Assoc. Comput. Mach., Vol.26, pp.494-526, 1979.

V. Klee and G. J. Minty. How good is the simplex method? Inequalities III. Academic Press,NY, 1972.

C. E. Lemke. On complementary pivot theory. In G. B. Dantzig and A. F. Veinott, editors,

Mathematics of decision sciences, Part 1, pages 95_114. AMS, Providence, Rhode Island, 1968.

Dantzig, G.B.: ”Linear programming and extensions”, Princeton University Press, 1963
Lineáris Programozás I, Bólyai János Matematikai Társulat, Budapest, 1968

Chen S.S., D.L. Donoho, and M.A. Saunders, Atomic Decomposition by Basis Pursuit,” *SIAM J.on Scientific Computing, 20*, 1, pp. 33-61, 1998.

Gill, P., W. Murray, M. Saunders, and M. Wright, "A Practical Anti-Cycling Procedure for Linearly Constrained Optimization," *Mathematical Programming*, 45, pp. 437-474, 1989.

Dr. Csernyák László, ”Operációkutatás II.”, Nemzeti tankönyvkiadó Rt 1999.

The simplex method:
www-p.mcs.anl.gov/otc/GUIDE/OptWeb/continuous/constrained/linearprog/section2_1_1.html

retroLP, AN IMPLEMENTATION OF THE STANDARD SIMPLEX METHOD:
http://cis.poly.edu/rvslyke/implmnt.pdf

Lp feladatok leírása:
www.stud.u-szeged.hu/Jenei.Peter.2/jegyzetek/opkut.doc

Lineáris Programozás:
http://www.bke.hu/puskas/elibd/szimplex.pdf

Bajalinov Erik, Rácz Anett, ”Skálázási folyamatok értékelése”
http://www.inf.unideb.hu/~fattila/hallgatoknak/dokumentumok/Racz_Anett.pdf

Komáromi Éva, ”Operációkutatás”, 2005
http://web.uni-corvinus.hu/~opkut/elibd/linearis_programozas.pdf

MPS formátumról:
http://www.hit.bme.hu/~lakatos/mps_format.txt

MPS file format:
http://lpsolve.sourceforge.net/5.5/mps-format.htm

MPS input format:
http://www-fp.mcs.anl.gov/otc/Guide/OptWeb/continuous/constrained/linearprog/mps.html

MPS (format):
http://en.wikipedia.org/wiki/MPS_(format)

The MPS file format:
http://www.mosek.com/products/3/tools/doc/html/toolbox/node10.html

MPS format:
http://www.sci.hkbu.edu.hk/msc/full/doris/node19.html

The classical Simplex Method
http://home.ubalt.edu/ntsbarsh/opre640a/partIV.htm

The Revised Simplex Method
http://www.cise.ufl.edu/~davis/Morgan/chapter2.htm#rs

Simplex method
http://www.math.cuhk.edu.hk/~wei/lpch3.pdf