Debreceni Egyetem

Informatika Kar

Operációkutatási feladat megoldása saját fejlesztésű Borland Delphi program és Lingo dll segítségével

Témavezető:

Dr. Bajalinov Erik

Tudományos főmunkatárs

Készítette:

Péterfy Zoltán

Programozó matematikus

Debrecen

2008

Tartalomjegyzék

1. Bevezetés

Diplomamunkám és kutatásom célja, hogy készítsek egy olyan alkalmazást, amely segítségével bárki, aki nem ismeri a Lingo/Lindo programcsalád speciális szintaktikáját, könnyen meg tudjon oldani lineáris egyenletrendszereket és lineáris programozási feladatokat. Célom, hogy feltérképezzem, milyen pontokon és hogyan lehet a LINGO megoldó motorjához csatlakozni, majd azután csak a LINGO dll-jeit felhasználva, egy gyors és felhasználóbarát felületet biztosítani.

Diplomamunkámban szeretném bemutatni a Lindo programcsomagjának néhány jellemzőjét, a LindoApi - t és Lingo - t, mint „black box" - ot. A programom segítségével az operációkutatásban igen elterjedt MPS formátumban megadott optimalizálási feladatokat lehet betölteni, megoldani, és .MPS kiterjesztésű állományokba kimenteni a feladatot. Fontosabb, az alap lineáris programozási feladaton túlmutató feladat is előállítható, ezért tárgyalni fogom részletesen az MPS – formátumot (Mathematical Programming System), a lineáris programozást, mint matematikai – közgazdasági szakterületet (Linear Programming), továbbá írni fogok a Delphi programozási nyelvről,mint fejlesztőeszközről .

Kutatásom során azért főleg lineáris programozási feladatokkal foglalkozom, mert ezek a fajta feladatok leggyakoribb operációkutatási problémák. A lineáris programozási feladatok nagy részének igen egyszerű és triviális megoldása van, de azért igyekeztem olyan alkalmazást fejleszteni, mely képes lesz bonyolultabb egyenletek , egyenletrendszerek megoldására. Bár a Lingo modellezés során tudunk hasznos beépített függvényeket és Set-ek által mátrixokat is használni, itt kisebb átalakításokkal lehet majd ezeket szimulálni.

Az operációkutatás (operational research) problémáinak megoldásával és implementálásával sok nemzetközi cég foglalkozik, így az interneten keresgélve hamar találhatunk jó néhány Solver -t, de én elolvasva Kiss Viktor 2007 -es szakdolgozatát (amelyben a Solverek gyorsaságát és bonyolultságát tesztelte) és saját tapasztalatimra hagyatkozva a LINGO 10-es verzióját választottam.

De a teljesség nélkül a legnépszerűbbek:

- LindoApi
- CPLEX
- CoinMP
- GLPK (Gnu Linear Programming Kit)
- LPSolve
- LGO
- Conopt
- Conopt2

Ezen Solverek általában komoly és költséges fejlesztések végeredményei, ezért a felhasználók csak, mint fekete dobozként láthatják az eszközöket, mert a piaci versenynek megfelelően a megoldások szuper titkos „know-how" -nak tekinthetőek. Pontosan ebből következik, hogy általában licenc kötelesek, próbaverziók 1-2 hónap kipróbálási időtartamra ingyenesek. Hiszen gondoljunk csak bele, túllépve a mi néhány változós lineáris egyenleteinken, hogy mennyire bonyolult lehet egy világméretű Lufthansa konszern repülőgép hálózatának optimalizálása vagy olyan olajipari cégek, mint ExxonMobile vagy Shell gyártási kapacitása és szállítása.

Célom tehát egy saját fejlesztésű program megírása és dokumentálása Lingo Solver használatára Delphi 5.0 (Object Pascal) nyelv használatával, mert akik komolyan szeretnének később operációkutatási problémák megoldásával foglalkozni, azoknak nagy segítséget jelenthet ez az alkalmazás, hogy a folyamatosan fejlődő és változó programcsomaghoz egy jó kiindulási alap legyen.

2. Alapok

2.1. Lineáris programozás

A lineáris programozás főként azzal foglalkozik, hogy lehet szétosztani bizonyos szempontból optimálisan korlátozott forrásokat különböző tevékenységek között. A lineáris jelző arra utal, hogy a probléma megfogalmazásában lineáris függvények szerepelnek, a programozás pedig a tervezésre utal, és nem a számítógépes programozást jelenti.

A lineáris programozást és szimplex – módszert 1947-ben George Dantzig (1914 - 2005) fedezte fel, bár az alapokkal elsőként talán a szovjet közgazdász és matematikus Leonid Kantorovich foglalkozott. Kantorovich – ot 1975 – ben Közgazdasági Nobel – díjjal jutalmazták, egyedüliként kapta meg a Szovjetunió létezése alatt.

A következő leírást Rapcsák Tamás leírásából emeltem ki:

Az operációkutatatási problémák megoldása során először a gyakorlati probléma egy matematikai vagy matematikai formalizmussal megadott modelljét kell megalkotni, azután az abból adódó feladatokat megoldani. Az operációkutatás legjellegzetesebb modelljeiben a döntéselőkészítés folyamán optimalizálás történik, azaz vagy a ráfordítási költségeket kell minimalizálni vagy a nyereséget maximalizálni a technológiai és/vagy egyéb kötöttségekből adódó feltételek teljesülése mellett.

Az ilyen típusú problémák megoldása a matematikai programozás alapfeladata. A matematika nyelvén megfogalmazva ez azt jelenti, hogy egy többváltozós függvény maximumát vagy minimumát keressük a tér egy részhalmazán. Ez a részhalmaz, illetve az ún. célfüggvény a megoldandó probléma jellegzetességeiből adódóan különböző tulajdonságokkal rendelkezhet. Ettől függően kapjuk a matematikai programozás különböző ágainak az alapfeladatait.

A matematikai programozási feladatokban az ún. feltételi halmaz általában egyenlőségekkel és egyenlőtlenségekkel van megadva. Ha feltételi függvényekben, (melyek a feltételei halmazt definiálják) és a célfüggvényben szereplő legalább egy diszkrét vagy egy valószínűségi változó, akkor beszélhetünk folytonos, vagy determinisztikus, illetve diszkrét vagy sztochasztikus programozási feladatokról. A folytonos és a többi matematikai programozási feladatosztályon belül a feltételi függvények és a célfüggvény típusától függően lehetnek lineáris, lineáris feltételrendszerű, kvadratikus, szeparábilis, geometriai, konvex, általánosított konvex, általános nemlineáris stb. programozási feladatok.

Jelenlegi ismereteink szerint nem létezik olyan univerzális algoritmus, amelyik minden feladatot megoldana, sőt az egyes speciális feladattípusokon belül olyan komoly nehézségek léphetnek fel, amelyek lehetetlenné teszik még a feladat közelítő megoldásának megkeresését is. Ez magyarázza a nagyszámú algoritmus létrejöttét és az implementálás, valamint az experimentálás jelentőségét.

2.2. Az MPS adatformátumról

Egy kicsit részletesebben szeretném bemutatni az MPS (Mathematical Programming System) formátumrendszert, több okból is. Egyrészről ez egy jól felépített, szabványosított formátum, amit a LINGO és egyéb más Solverek is értelmezni illetve feldolgozni tudnak. Másrészről az általam készített Delphi alkalmazás is képes a felhasználó által begépelt egyenleteket ilyen formátumban letárolni. Nem kis erőfeszítésembe telt, hogy ezt a funkcionalitást implementáljam, hiszen elég szigorú szabályokat kellett kielégítenem. A beolvasás egy fokkal könnyebb volt, de ott is akadtak problémák, főleg a BOUNDS szekcióban, de ne szaladjunk előre.

Általánosan megfogalmazva az MPS egy olyan formátum, amely lineáris programozási feladatok és kevert egész értékű (mixed integer) programozási feladatok tárolására és archiválására hoztak létre. Az MPS formátumot egy korai IBM program után neveztek el, és vált gyakorlatilag ASCII szabvánnyá.

Az MPS oszlop-orientált formátum, ami azt jelenti, hogy nem egyenletekként kell megadnunk a modellt. A modell komponensei (változók, sorok) neveket kapnak. Az MPS - t még lyukkártyákra tervezték ezért nem szabad-formátum, azaz rendkívül szigorú és kötött minden egyes karakter helye és jelentése. Az MPS -ben megadott feladat szegmenseinek elnevezései vannak, és ezek az elnevezések a szegmens első sorában kapnak helyet. Bár tipikusan mindenhol mindent nagybetűvel kell írni (én is ezen konvenciót követtem az alkalmazásomban) a formátum történelmi okai miatt, de napjaink MPS olvasói/értelmezi elfogadnak bármilyen kis és nagybetűs MPS állományt is, illetve vannak olyanok, ahol megkövetelik, hogy a szegmensek elnevezései csupa nagybetűsek legyenek, de egyébként pedig elfogadja a kis és nagybetűket is bárhol máshol. A nevek, amelyek megadhatóak az egyes változóknak illetve megszorításoknak a Solver szempontjából lényegtelenek, tehát igyekezzünk értelmes vagy egyszerű nevet adni nekik a későbbi olvasás és szerkesztés megkönnyítése miatt. Itt szeretném megjegyezni, hogy bár Delphiben elég szabadon hagytam a változók elnevezését és ez által a hosszát is, azért célszerű, maximum 8 karakter hosszan megadni, mert később az MPS fájlba mentésnél csak az első 8 karakterét fogja kimenteni.

A szabvány szerint 5-12 karakterek tartalmazzák MPS formátumban a változók nevét COLUMNS szekcióban.

Egy probléma megadása MPS formátumban:
Nézzük meg a különbséget a normál lineáris programozási feladatot normál egyenletrendszerként, illetve MPS formátumban felírva.

```
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
```

Míg az elvárt forma:

Optimize
 COST: XONE + 4 YTWO + 9 ZTHREE
Subject To
 LIM1: XONE + YTWO < = 5
 LIM2: XONE + ZTHREE > = 10
 MYEQN: - YTWO + ZTHREE = 7
Bounds
 0 < = XONE < = 4
-1 < = YTWO < = 1
End

2.3. Az MPS kulcsszavai és szekciói:

- NAME: Itt lehet megadni a lineáris programozási feladat nevét, a feladat megoldása szempontjából a név lényegtelen. Igazából a LINGO is csak Title és név párossal olvassa be, de kihagyja az értékelését. Csak az első sorban szerepelhet, a név érték 15-22 – ig van letárolva, azaz ez is ad egy megszorítást.

ROWS: Itt találkozunk először a sorok elnevezésével, (OBJ jelöli a célfüggvényt). A ROWS négy karakter, abban a sorban már semmi nem lehet. A következő sor a célfüggvény azonosítása, 'N' betűvel kezdődik és OBJ kulcsszó áll 5 – 12 – ig tartó helyen. A szekció további sorai azonosítják az egyenleteink további neveit illetve a relációs jeleket. E-vel lehet megadni az egyenlőségi feltételt, L-el a kisebb-egyenlő relációt, G-vel a nagyobb-egyenlő relációt.
A ROWS szegmensben a sorok sorrendje a probléma megoldásának szempontjából lényegtelen. Csak az adott szekcióban bevezetett azonosítókkal rendelkező sorok szerepelhetnek a ROWS szekció után következő szekciókban!

- COLUMNS: Ebben a szegmensben van megadva minden egyes változónak a szorzója, hogy a célfüggvényben illetve a megkötésekben az egyes változók milyen együtthatóval szerepelnek.
Természetesen, ha egy változó egy feltételben nem szerepel, vagyis nullaszor van benne, akkor az itt nincs külön megadva. (De lehetséges egy változót nullával megadni.) Tulajdonképpen ebben a szegmensben van megadva a feladat A mátrixa, amiből az is következik, hogy a jobboldali „B” vektor együtthatói nem itt vannak letárolva. A sorok sorrendje a feladat megoldásának szempontjából nem fontos. Összegezve tehát a szekció COLUMNS kulcsszó áll az első sorban, majd alatta felsorolva a változók nevei (5-12), melyik sorban szerepelnek (15-22), milyen együttható értékkel szerepel a modellben (25-36). Itt kell megjegyezni, hogy ha ezen sorban szereplő változó az egyenletrendszer más sorában is szerepel, akkor lehetséges az MPS fájl ezen sorában még egy modell előfordulási sor (40-47) és érték (50-61) pár megjelenítése.

- RHS: Ebben az egységben tudjuk megadni a jobb oldali „B” vektort (Right-Hand-Side). Egy vagy több vektor is megadható, egynél többet csak nagyon ritkán adnak meg. Több jobboldali vektor esetén meg kell adni, hogy a feladat megoldása során melyik jobboldali vektort kell használni. A szegmensben a jobb oldali vektor el van nevezve, és amely megkötésére nincs érték megadva, ott alapértelmezésként 0 szerepel a jobb oldali vektorban.
Általános alakja: RHS kulcsszó, majd abban a sorban semmi más nem állhat. 5 -12 karakterek határozzák meg a fent említett RHSVektor nevét, majd a 15 – 22 helyen letárolt karaktersorozat reprezentálja a sor megnevezését, amelyre szintén igaz, hogy csak olyan lehet, amely már ROWS szekcióban definiálva volt! Az értéket 25 – 36 -ig terjedő karakterekből nyerhető ki.

- RANGES: RANGES: Ebben a szegmensben lehet a ROWS szegmensben megadott kisebb-egyenlő és nagyobb-egyenlő relációihoz intervallumot hozzárendelni. Tehát amelyik megkötésnek a RANGES szegmensben intervallumot adnak, annak így meg van adva az alsó és a felső korlátja is. Legyen a megkötésünk jelülése Ci, az RHS szegmensben hozzárendelt érték Bi, és a RANGES szegmensben hozzárendelt érték Ri. Ekkor, ha Ci „<=” reláció, akkor az értéke (Bi-| Ri |) <= Ci <= Bi tartományba esik. Ha pedig Ci „>=” reláció, akkor az értéke Bi <= Ci <= (Bi +| Ri |) tartományban kell legyen. A RANGES szegmens opcionális, nem kell benne lennie az MPS állományban, ha nincs rá szükség. A lineáris programozási feladatok nagytöbbségében nincs RANGES szegmens. A magyarázat után következzen a formális leírás: A szekció természetesen a RANGES kulcsszóval kezdődik és abban a sorban már semmi más nem szerepelhet. Az értékeket reprezentáló sorok az RHS -hez hasonló (5 - 12) <RANGESVektor>,(15 - 22) <sor>,(25 – 36) <érték> hármasból épülnek fel.

- BOUNDS: Egy nagyon érdekes és izgalmas szegmens, de is opcionális. Ebben a szegmensben a lineáris programozási feladat változóinak értékeire vonatkozó korlátokat adjuk meg. A LINGO és szimplex – módszer megkötései szerint egy változó $0 <= X_i$. Azaz minden X_i „0” és „+ végtelen” között vehet fel értékeket. Ezt az értéktartományt terjeszthetjük ki a negatív tartományba, vagy korlátozhatjuk a változó értékét valamilyen R -beli alsó vagy felső korlátra. Speciális estben bináris (@Bin(x)) és egész értékű (@Gin(y)) megszorítást is

alkalmazhatunk. Általános alakja : 2-3 pozición egy két karakter hosszú kód található (magyarázatot lásd a leírás alatt), 5-12 -ig a Bound neve, 15 – 22 -ig a változó, amelyre a kikötést alkalmazzuk, 25-36 -ig az értéket tároljuk, már ha van értelme, ugyanis ha bináris lesz akkor a kulcsszóból már tudjuk az értékeket.

A Bounds típusai:

	type	meaning
LO	lower bound	b <= x (< +inf)
UP	upper bound	(0 <=) x <= b
FX	fixed variable	x = b
FR	free variable	-inf < x < +inf
MI	lower bound -inf	-inf < x (<= 0)
PL	upper bound +inf	(0 <=) x < +inf
BV	binary variable	x = 0 or 1
LI	integer variable	b <= x (< +inf)
UI	integer variable	(0 <=) x <= b
SC	semi-cont variable	x = 0 or l <= x <= b

l is the lower bound on the variable

If none set then defaults to 1

- ENDDATA: Kötelező kulcsszó, amely lezárja az MPS állományunkat. Természetesen ebben a sorban csak ez a kifejezés állhat.

3. A LINDO-ról

A LINDO (http://www.lindo.com) egy Chicago – i székhelyű vállalat, amely 21 éve kínál megoldásokat a gazdasági élet vezető szereplőinek. A saját állításuk szerint piacvezetők a gyors és egyszerű matematikai optimalizáció területén. Lineáris programozási (linear programming), egész értékű programozási (integer programming), nem-lineáris programozási (non-linear programming), kvadratikus programozási (quadratic programming) termékeit a világ TOP 25 cégéből 23 alkalmazza. Például a bevezetőmben említett: Exxon Mobil .
Szerencsére a termékükhöz 1 próba hónapra adnak licence – t, de támogatják az egyetemi oktatást is, hiszen egyetemek és hallgatók rendelkezésére bocsátanak „lebutított" verziókat, oktatási licencekkel. Azaz a változók értékére megadnak korlátozásokat, de a mi szintünkön ez alig észrevehető.
A cég honlapján egyébként látható, hogy keresnek erős C/ C++ tudással rendelkező fejlesztőket, tehát a lehetőség mindenki előtt nyitva áll.
Például:

Software Developers
LINDO Systems is looking for exceptional candidates to play an integral part in the development of new optimization algorithms and user interfaces. These individuals will be involved in challenging projects such as designing GUIs, extending our modeling systems, and developing native APIs to the callable library from various languages. The ideal candidate will have a BS or MS in computer science, or a related area, with a strong background in data structures and algorithms, strong analytical abilities, and exceptional programming skills. Experience in C is a must.

A kis kitérő után térjünk vissza LINGO és LindoApi jellemzéséhez.

A LINGO programcsomaggal volt szerencsém Dr. Bajalinov Erik tanár úr gondozásában tartott órákon megismerkedni. A kezdeti nehézségek (speciális nyelvi sajátosságok és modellezési stílus) után, a Solver rögtön bebizonyította, hogy gyors és megbízható. Míg papíron egy szimplex-módszer megoldása több 10 percet vesz igénybe, egy jóval bonyolultabb és komplexebb feladatok modellezése, megoldása és a LINGO Riport

kiértékelés töredéke mindezeknek. Nem is beszélve szállítási feladatok (transportation problem) vagy hálózati feladatok (networking problem) megoldásának.

A Lingo és Lindo letöltésekor automatikusan a birtokunkba kerül egy több száz oldalas leírás és a LindoApi is amely magában hordozza azt a jó tulajdonságot, hogy a legtöbb ismert programozási nyelvvel (VB, Java, C/C++, Delphi, .NET, FORTRAN) csatlakozási pontot létesíthetünk vele és válaszként a helyes optimalizált megoldást adja vissza. Továbbá mellékelnek külön példaprogram kódrészleteket is amellyel szemléltetik, hogy is kell a két rendszer közti interfészt létrehozni. Egyébként az egyik ilyen speciális kapcsolat az MPS fájl, de én másikat alkalmaztam.

4. Delphi és a Delphi 5-ös verzió

4.1. A Delphiről általánosságban

A Windows egész filozófiája az ablakokra van alapozva, ezért a Windowsos felületre tervezett Delphi alkalmazásokat is ablakokban (formokban) kell elképzelni. A program (projekt) különböző részei, műveletei egy-egy, arra a feladatra tervezett ablakban játszódnak le. Azaz a program felhasználói felületét az ablakok szolgáltatják. A Delphiben kétfajta programozási stílussal dolgozhatunk. Az egyik a komponensekkel való programozás, amikor a komponensek előre elkészített építőelemek. A másik programozási mód a „hagyományos kódolás", a problémamegoldás Object Pascal programnyelven való megfogalmazása. A delphi a forráskód egy részét saját maga készíti el, a tulajdonképpeni algoritmusokat nekünk kell megírnunk.

A 4GL eszközök magas szintű programozási nyelvre épülő komplex, objektumorientált programfejlesztői rendszerek. A 4GL rendszerek a hagyományos programozási nyelvektől eltérően nem igénylik az összes elemi -különösképp a felhasználói felületekre vonatkozó- tevékenységek részes implementációját. A programozó készen kap olyan elemeket, objektumokat, amelyekkel ezeket a gyakran időrabló feladatokat gyorsan és hatékonyan képes megoldani. Így például nem szükséges a felhasználói felületet a részletek szintén utasításonként programozni, elegendő csak a képernyőn megjelenő elemeket megadni, és a fejlesztőrendszer ez alapján már automatikusan generálni tudja az űrlap (ablak) egy alapértelmezés szerinti felépítését. Ezzel a módszerrel a 4GL eszközökkel történő fejlesztés során a programozó magasabb szintű absztrakcióval készítheti a programjait. Erre az úgynevezett komponens alapú fejlesztési módszer teremti meg a lehetőséget. A komponensek olyan előre gyártott építőelemek, melyeket a fejlesztőkörnyezet tartalmaz. A programozó ezekből a vizuális tervezőfelületen összeállíthatja a programja vázát, felhasználói felületét. Ezen a szinten a programkód is automatikusan generálódik, tehát a fejlesztő idejének nagyobb hányadát fordíthatja a speciális algoritmusok (alkalmazás logika) elkészítésére. Manapság a nagyobb rendszereket szinte kizárólag ilyen 4GL környezetben fejlesztik és kisebb

alkalmazásoknál is egyre jobban teret hódít ezek használata. A különböző 4GL rendszerek eltérhetnek egymástól az egyes funkciók tekintetében és azok lehetőségeiben is.
Nincs egységes követelmény arra nézve, hogy mit és milyen szinten kell tartalmaznia egy 4GL rendszernek, meg lehet azonban határozni olyan jellemzőket, amelyek általánosan minden 4GL eszközre vonatkoznak.

Ebbe a csoportba tartozik a Delphi is. (vagy például a Visual Basic) amely általános programozási feladatok megoldására szolgál. Azért a technológia meghatározásánál mindig figyelni kell a feladat megoldásának igényére, hiszen nem biztos, hogy mindig a legerősebb eszközre van szükségünk, gondoljunk az eszköz hardver igényére, illetve a kollégák szaktudására.

A Borland Delphi rendszer olyan, gyors alkalmazásfejlesztő környezet (RAD Rapid Application Development), amely hazánkban rendkívüli népszerűségnek örvend. A Delphi (és testvére a Kylix Delphi 7-ben) már szerencsésen egyesíti magában a Windows és Linux alkalmazások készítésére szolgáló grafikus fejlesztő környezetet és a teljesen objektum orientált programnyelvi fordítót. A rendszer továbbá támogatja a legújabb technológiákat, úgymint Internet-alkalmazások, multimédiás megjelenítés és ügyfél kiszolgáló adatbázis – kezelés.

A Delphi alkalmazások általában több ablakot (formot) tartalmaznak. Az ablakok közül általában egy adja az alkalmazás menüvel ellátott főablakát, míg a továbbiak párbeszéd- vagy gyermek ablakok, melyek menüpontok kiválasztásakor, vagy nyomógombok lenyomásakor jelennek meg. Bár ettől az ajánlástól eltér a saját fejlesztésű programom, de fogadjuk el, hogy ez az etalon.

Az újrafelhasználás vagy kódelrejtés (például fekete doboz létrehozása) miatt, fogalmazódik meg az igény, hogy a jól megírt, letesztelt programrészek egy következő hasonló fejlesztési feladatnál ismét elérhetőek legyenek, rendelkezésünkre álljanak. Ezt csak akkor végezhetjük el, ha úgy alakítjuk ki a program különböző részeit (alprogramokat, speciális eljárásokat, függvényeket), hogy azok továbbvihetőek legyenek. A Delphi erre is kínál megoldást, a dinamikusan szerkeszthető könyvtárak (DLL Dynamic Link Library) és komponensek

segítségével. A DLL – ek a létrehozásuk után bármikor tetszőlegesen felhasználhatóak illetve más fejlesztői környezetben (például: Microsoft Visual Basic, Microsoft Visual C++) is alkalmazhatóak. De ez természetesen fordítva is igaz, így tudtam a Lingo fejlesztésű dll – fájlt (Lingd10.dll) az applikációmban meghívni (stdcall) majd a megoldást visszakapni. Tehát vegyük észre, hogy a DLL – ek fejlesztése milyen fontos, de ugyanakkor könnyű és hatékony eszköz a Delphi fejlesztők palettáján.

A Delphi több lehetőséget kínál a háttértárakon tárolt adatok írására és olvasására. A kivitelezésre használhatjuk az Object Pascal eszközeit, illetve a Windows rendszer fájlműveleteit. Nagyon könnyen és egyszerűen tudtam az alkalmazásom során a LoadFromFile() és SaveToFile() műveleteket megvalósítani illetve a beolvasott állományokat szekvenciálisan, soronként feldolgozni. Ez is kényelmes, jól implementált eszköz. Végül a fejlesztőeszköz bemutatását szeretném még egy dicsérettel lezárni, hiszen sokan a Delphit, mint az adatbázis és Windows közti legjobb és Magyarországon is elég népszerű programozási nyelveként ismerik. Valóban gyorsan és könnyen lehet vele adatbázis-kezelési műveleteket és felhasználói felületeket létrehozni, az adatbázis kapcsolata könnyű és gyors, köszönhető ez a BDE (Borland Database Engine) motornak, kisebb vállalkozásoknak tökéletes adatbeviteli felületet és tárolási megoldást tudunk biztosítani.

4.2. Az alkalmazásom és a Lingo közti interfész

A megvalósításhoz a Lingo cég is nagyban hozzájárul, hiszen ingyen segítségként adja a Delphi 5 – höz teljesen illeszkedő Lingd10.dll –t és a Lingd10.pas állományt. A fájl tartalmazza az interfész összes funkciójának prototípusát a Lingo működéséhez szükséges konstans változókat. Ezután ha a főablakunk .pas részébe, beírjuk a következő sort, akkor azonnal elérhetővé és láthatóvá válik a DLL.

uses lingd10;

Az interfész megvalósításához is kapunk segítséget, egyrészt példa kód és a Lingo10 dokumentációja (Lingo Users Manual.pdf 676 oldal !) alapján kézenfekvő a megoldás:

```
//  LINGO callback implementáció
function  MyCallback ( pLingo: Integer; nReserved: Integer;
 nUserData : Integer): Integer; stdcall;

var
   nIterations : Integer;
   pedt : ^TEdit;

begin

   //levesszük az aktuális iterációs értéket
   LSgetCallbackInfoLng( pLingo, LS_IINFO_ITERATIONS_LNG, nIterations);

   // megjelenítjük az iterációs számlálót
   pedt := pointer( nUserData);
   pedt.Text := intToStr( nIterations);

   // return a nonzero value to halt optimization
   //
```

```
  MyCallback := 0;

end;
```

Azaz a végső megoldás csak néhány lépésre van. Szükségünk van az optimalizálni kívánt feladatra és feltételekre, amelyeket az alkalmazás formjairól leolvashatunk, szükségünk van a változókra amiket, majd mint mutatókat fogunk megadni, és ezeket összeszedjük egy .lng kiterjesztésű fájlba.

Még hátravannak a Lingo és Windows környezeti beállítások, de ez minden futáskor szekvenciálisan, a beállításfüggően hajtódik végre. Beállítjuk a Lingo környezetet:

```
pLingo := LScreateEnvLng();
```

Megnyitjuk a Log fájlunkat, amire azért van szükség, mert a Lingo „fekete doboz” ellenére, mi monitorozni szeretnénk a működést, illetve a hiba esetén szeretnénk tudni, miért volt sikertelen a megoldás. Tehát megnyitjuk a LINGO.log fájlt is. Természetesen visszatérési értékként megkapjuk, hogy sikeres volt-e a megnyitás.

```
nErr := LSopenLogFileLng( pLINGO, 'LINGO.log');
```

Ezek után átadjuk az interfészünkön keresztül, hogy milyen menő változóink lesznek. Itt kell megmondani az objektumot, azaz hogy mit is szeretnénk optimalizálni (1.lépés), az iterációs számot (2.lépés) illetve, hogy milyen változóink vannak. Itt minden olyan változót definiálni kell a lent látható formában, amelyiknek az értékére kíváncsiak leszünk a végén és az optimális megoldásban szerepet játszik (3.lépés).

```
// beállítjuk bemenő paramétereket
  cbf := MyCallback;
  nErr := LSsetCallbackSolverLng( pLINGO, @cbf, integer( @edtIterations));
  if ( nErr <> 0) then goto ErrorExit;
```

1.lépés:

```
nErr := LSsetPointerLng( pLINGO, dObj, nPointersNow);
if ( nErr <> 0) then goto ErrorExit;
```

2.lépés:

```
nErr := LSsetPointerLng( pLINGO, dStatus, nPointersNow);
if ( nErr <> 0) then goto ErrorExit;
```

3.lépés: (nem Delphi pszeudo-kód !)

```
for i=1 to  összes(aktuális_változó) do
begin
    nErr := LSsetPointerLng( pLINGO, aktuális_változó[i], nPointersNow);
    if ( nErr <> 0) then goto ErrorExit;
end;
```

Ha fentieket betartottuk, akkor a Lingo már értelmezni tudja, hogy milyen feladatot szeretnénk vele megoldatni. Ha összeállítunk, egy egyszerű karakterláncot a lenti Lingo ajánlás szerint, akkor tényleg egy lépésre vagyunk a megoldástól. A fenti említett .lng kiterjesztésű fájlunk neve legyen pelda.lng.

```
// Összerakunk egy futtató scriptet
cScript := 'SET ECHOIN 1' + Char( 10) +
 'TAKE pelda.LNG' + Char(10) +
 'GO' + Char( 10) +
 'QUIT' + Char( 10) +
 Char( 0);
```

Ténylegeses futtatás a következő paranccsal történik, ezért az egy sorért történik az alkalmazásban minden. Minden beolvasás, minden gombnyomás, minden beállítás:

```
nErr := LSexecuteScriptLng( pLINGO, cScript);
```

Ha semmi problémánk nem akadt, azaz nErr értéke „0” akkor fellélegezhetünk, mert a Lingo helyesen működöt és a megadott lineáris programozási feladatot értelmezte és megoldotta és már nincs szükségünk a Lingo.LOG fájlunkra, ezért bezárhatjuk.

LScloseLogFileLng(pLINGO);

Sajnos a helyes működés még nem mindig egyenlő a sikerrel, mert ha nincs a feladatnak optimális megoldása, akkor sajnos, habár minden az előírásoknak megfelelő volt, de a feladatunk nem megoldható. A beállítások 2.lépésénél (nem véletlenül részleteztem) adjuk meg paraméterként a dStatus változót, ha ez nem egyenlő a Lingo konstansként beállított értékéből, abból következik, hogy nem létezik optimális megoldás (1.eset).

Ha helyesen adtuk át a változóink nevét, és minden rendben volt, akkor viszont, nincs más hátra, mint megjelenítjük az optimális megoldás értékét, tetszőleges formában (2.eset).

Nézzük ezt, hogyan lehet implementálni:

```
if  ( dStatus <> LS_STATUS_GLOBAL_LNG)) then

    ShowMessage( 'Unable to solve to optimality.')

else

        //Megjelenítjük a helyes értékeket.
        //Például: az optimális megoldás a dObj – ben van letárolva:

        Results.Edit1.Text := floatToStr( dObj);
End; //if.
```

Nem szabad megfeledkezni a memória felszabadításról, mert sajnos nem automatikus, figyeljünk rá!

LSdeleteEnvLng(pLINGO);

Ezen lépések sorozatával lehet a Lingo interfészünket implementálni, de szükségünk van a .lng állományra, ezen fájl előállításával egy következő fejezetben fogok részletesen foglalkozni.

5. Az .lng állományról

Ha megnyitjuk a Lingo10 alkalmazásunkat, akkor láthatjuk, hogy milyen típusokat lehet beolvasni. Lehetőségünk van:

- Lingo Models (.ltx)
- Lingo Models (.lg4)
- Lingo Text Models (.lng)
- Lingo Data (.ldt)
- Lingo Script (.ltf)
- Lingo Report (.lgr)
- MPS Models (.mps)

kiterjesztésű fájlok beolvasására. A .lng Text modellnek is megvan a sajátságos felépítése, formai követelménye. Először nézzünk egy példát:

Példa a .lng fájlra:

```
model:
[obj] Max=(1)*x1+(23)*x2+(4)*x3;
4*x1+1*x3<=400;
2*x2<=200;
data:
@pointer(1)=obj;
@pointer(2)=@status();
@pointer(3)=x1;
@pointer(4)=x2;
@pointer(5)=x3;
enddata
end
```

Nézzük részletesebben. „model:” kulcsszóval indul és „end” zárja az általános Lingo formátumú modellt, igaz ez akár .lg4 de akármelyik verzióra is. Az [obj] kulcsszóval jelezzük, hogy ez a sor lesz az optimalizálni kívánt célfüggvény. A sorok végén mindig pontosvessző (;) áll, kivétel a kiemelt kulcsszavak. Természetesen a Max és Min szavak jelentése egyenlő azzal, hogy maximalizálni vagy minimalizálni szeretnénk az alapfeladatot. Az egyenletek

megadásánál ügyelnünk kell arra, ha az együtthatók negatív számok, bár a Lingo értelmezi a következő kifejezést is x1+-4*x2; de azért jobb az együtthatókat zárójelezni. Lingoban lehetőségünk van, a bounds szekcióban, a változóinkat kiterjeszteni, azaz nem egész értékűek hanem tört értékű értékeket is felvehetnek a változóink. A tizedes törtek jelölésére szolgáló karakter a pont (.) és nem a vessző. Ha közvetlenül a Lingonak adjuk meg a lineáris programozási feladatot, akkor erre figyelnünk kell. A saját fejlesztésű programomban ezt úgy oldottam meg, hogy az applikáción belül bármi szabadon használható, akár vessző akár pont tetszőlegesen, de a feldolgozás során a vesszőket pontra cserélem le.

A célfüggvény után következnek a feltételek. Tetszőlegesen sok feltételt fogalmazhatunk meg. Az egyenletek alakja Summa (A_{ij}*X_j) <= B_i ; Lehetőségünk van egyéb feltételek megadására is. A változók alapértelmezetten: X_j >= 0 értéktartományban vannak definiálva, de ezek is felüldefiniálhatók és sok feltétel megadható még például akkor → ha feltétel, vagy – vagy feltétel .

A fenti .lng fájlban megfigyelhető, hogy „data:" „enddata" szekcióban megadhatjuk a lineáris programozási feladat változóit, a következő formában :

@pointer(érték) = változónév ;

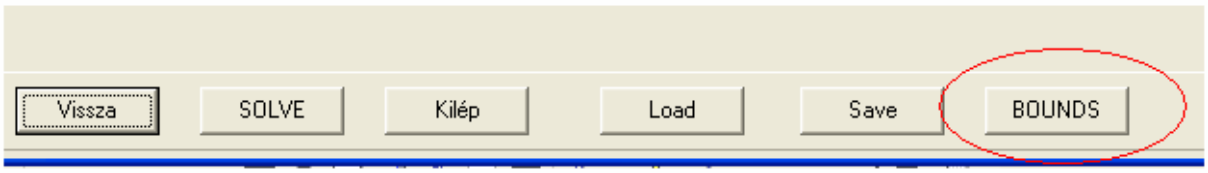
Ennek a fontossága abban rejlik, hogy az itt megadott változók, a megoldás után az interfészen keresztül kommunikáló programok részére visszaadja ezen értékeket. Így tudtam a programomban az optimális megoldást, a dStatus –t és az összes változó értékét visszakapni a futás után.

6. Az MPS implementálása

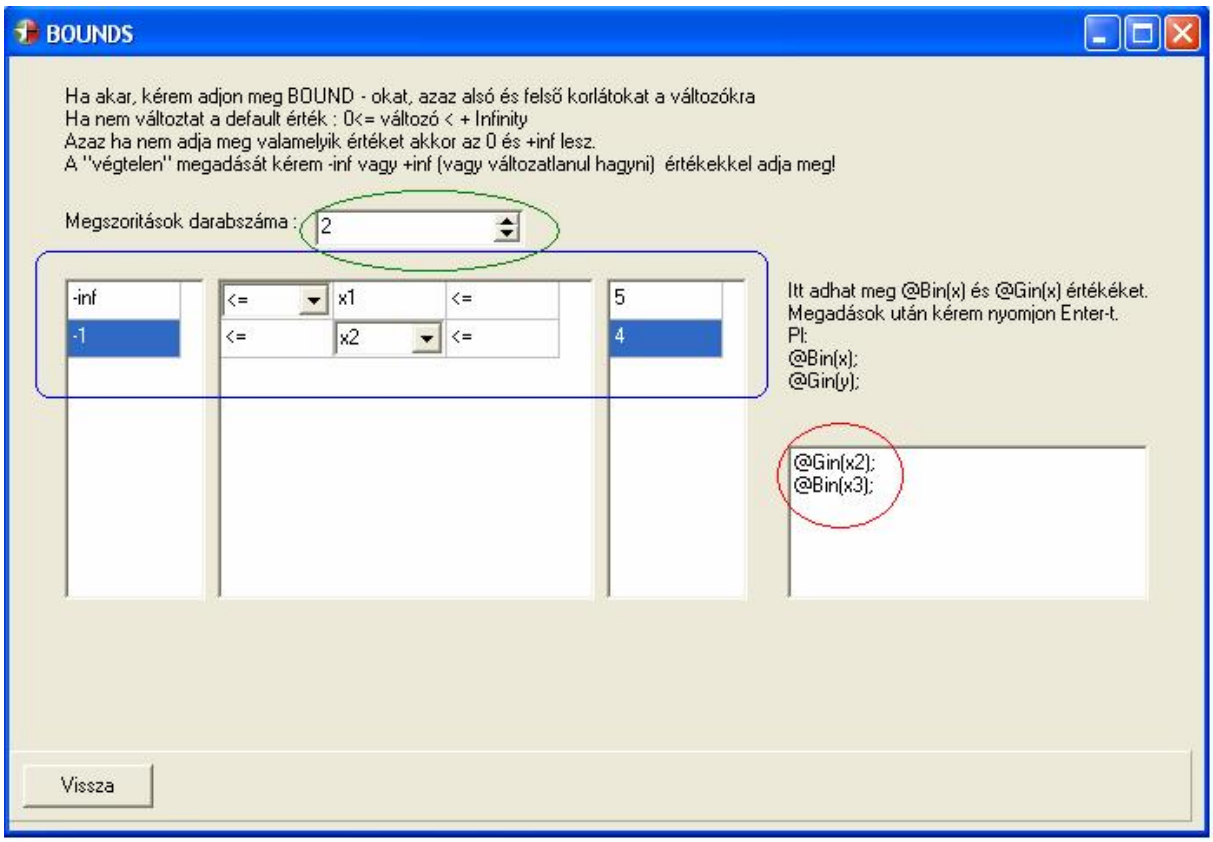
Az előző alapozó fejezetekben már bemutattam az MPS fájlformátum egyéni, speciális és nagyon kötött helyi értékű specifikációját. Most nézzük meg, hogyan is lehet ezt implementálni.

A kód írás közben állandóan néznem kellett a az előző fejezetekben már bemutatott táblázatot, mert a Bounds szekció MPS meghatározásában folyamatosan vannak átfedések.

Kezdjük az MPS formátumba mentéssel. A mentés funkció előtt a felhasználónak lehetősége van megadni a célfüggvényt, a feltételeket és a külön BOUNDS gombot megnyomva, aktiválhat egy ablakot, amely csak a megszorítások miatt hoztam létre. Az ablakot a Beviteli képernyőről az alábbi módon lehet elérni:



Kattintunk a fent kijelölt „BOUNDS” feliratú gombra és máris a következő ablakot látjuk:



A zölddel bekarikázott SpinEdit mező segítségével lehet, a megszorítások számát beállítani.
A kék lekerekített téglalap határolja, azt a területet, ahol megadhatjuk lenyíló menük segítségével, hogy melyik változót, milyen alsó és felső korlátokkal akarunk ellátni. Ha nem adunk meg semmit, akkor az alapértelmezett ($0 \leq x_j \leq +\infty$) feltétel lesz igaz a változókra. Ha ezeket felüldefiniáljuk, akkor azok lesznek a mérvadóak, azaz LO vagy LI kulcsszóval fognak szerepelni a .MPS fájlban, ha beírtunk valamit alsó korlátnak és UP vagy UI ha felülről korlátosnak akarjuk az értéket. Figyelni kell, mert ha beállítjuk a $-\infty$ -t alsó korlátnak, akkor szabad (free(változónév)) változóként fogja a Lingo kezelni, azaz se LO,LI se UP,UI nem szerepelhet, csak FR kulcsszóval állhat.

A pirossal bekarikázott részben lehet a @Bin(változónév) és @Gin(változónév) alakban megadni azokat a korlátozásokat, ha azt szeretnénk, hogy a változónk 0 vagy 1 értékű legyen (binary) illetve, hogy csak egész értékeket vegyen fel (general integer). Ezek nagyon fontos feltételek lehetnek, mert ha egészértékű megoldást szeretnénk (nem lehet 0.75 db autót , vagy 0.13 benzinkutat legyártani vagy 0.47 kórházat építeni, ha ez a feladat). Bináris változókkal, pedig azt szoktuk szimulálni, hogy kell vagy nem kell szerepelnie adott változónak a lineáris programozási feladat aktuális feltételében.

Ha betöltünk egy MPS formátumú fájlt, akkor ROWS, COLUMNS, RHS szekciók egyértelműek, bár azért az implementálásnál hasznos, ha dinamikus tömböket használunk, de a feldolgozásuk szekvenciális és könnyen algoritmizálható. Legalábbis könnyebben mint a Bounds – ok. Figyelni kell, hogy éppen melyik változót dolgozzuk fel, volt-e már előző sorokban hivatkozás rá. Például, ha egy változó alulról és felülről is korlátos, akkor az két sorban lesz letárolva.

LO BND1	*YTWO*	*-1*
UP BND1	*YTWO*	*1*

Ami természetesen azt fogja jelenteni, hogy $-1 \leq YTWO \leq 1$.
Ha adott változóról megkapjuk azt az információt, hogy a korlátja UI vagy LI akkor még pluszban a @Gin(változónév); feltételt is fel kell venni az ábrámon pirossal bekarikázott

részbe. Az FR és MI esetén az alsó korlát -∞ lesz. BV kulcsszó esetén „@Bin(változónév) ;” kifejezés fog szerepelni a pirossal jelölt szekcióban.

7. Egy érdekes probéma megoldása Delphi nyelven

Témavezetőmmel, Dr. Bajalinov Erikkel a feladat és felhasználói felület specifikálásakor, megbeszélünk egy – két beállítást, ablakok képét, gombok helyét, és természetesen a gombok által vezérelt eseményeket. Külön kérése volt, hogy relációs jelek kiválasztásakor, ne kézzel, hanem egy lenyíló listából lehessen választani. Először nem tudtam, hogy hozzákezdeni és az internet sem szolgált teljes megoldással, csak ötleteket adtak. Ezért szeretném a teljes és láthatóan jól funkcionáló megoldást itt publikálni.
Tehát adott egy StringGrid objektum és futási időben szeretnénk, ha bizonyos mezőjébe kattintva egy lenyíló listából (ComboBox) lehetne az értékeket kiválasztani.
Első lépésben létrehozunk a formon egy „láthatatlan" (visible := false) ComboBox - ot, aminek vagy az Items soraiba felvesszük (azaz beégetjük) az értékeket, vagy futási időben feltöltjük ezt a listát. A lista elemei String típusú értékek lehetnek, azaz karakterláncok.
A form létrehozásakor célszerű a StringGrid celláinak hosszát a ComboBox szélességéhez állítani, a szebb kivitelezés miatt, de ez tényleg csak a GUI design miatt.

```
procedure TForm.FormCreate(Sender: TObject);
begin
    StringGrid.DefaultRowHeight := ComboBox.Height;
end;
```

Az aktuális StringGrid DrawCell metódusát kell implementálnunk, a helyes megoldáshoz. Ha belekattintunk a mezőbe, akkor a DrawCell nagyon sok fontos információt átad a hívó eljárásnak. A cella sorát, oszlopát, a cella elhelyezkedését a Formon, „pixel" pontossággal.
De nézzük a kódot abból könnyebb lesz a megértés. A végén természetesen a jól pozícionált ComboBoxot megjelenítjük.

```
procedure TForm.StringGridDrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
  var
    R: TRect;
  begin
```

```
   if (ACol >= StringGrid.FixedCols) and
     (ARow >= StringGrid.FixedRows) and
     (gdFocused in State) then
     with ComboBox do
     begin
       BringToFront;
       CopyRect(R, Rect);
       R.TopLeft :=     Form.ScreenToClient(
                    StringGrid.ClientToScreen(R.TopLeft));
       R.BottomRight := Form.ScreenToClient(
                    StringGrid.ClientToScreen(R.BottomRight));
       SetBounds(R.Left, R.Top, R.Right-R.Left, R.Bottom-R.Top);
       ComboBox.Visible := True;
     end;
end;
```

Majd ha kiválasztottuk az értéket, akkor beírjuk cellatartalomnak és levesszük a ComboBox láthatóságát.

```
procedure TForm.ComboBoxChange(Sender: TObject);
begin
      with StringGrid do
      Cells[Col, Row] := ComboBox.Text;
      ComboBox.Visible := False;
end;
```

Ezzel a kis fejezettel szerettem segítséget nyújtani azoknak, akik StringGrid – e szeretnének lenyíló listát illeszteni, remélem, ezután sokan fogják használni.

8. A saját fejlesztésű programom bemutatása egy példán keresztül

Adott következő célfüggvény:

$3 * x1 + 2 * x2 + x3 \rightarrow max.$

A feltételek, pedig a következők:

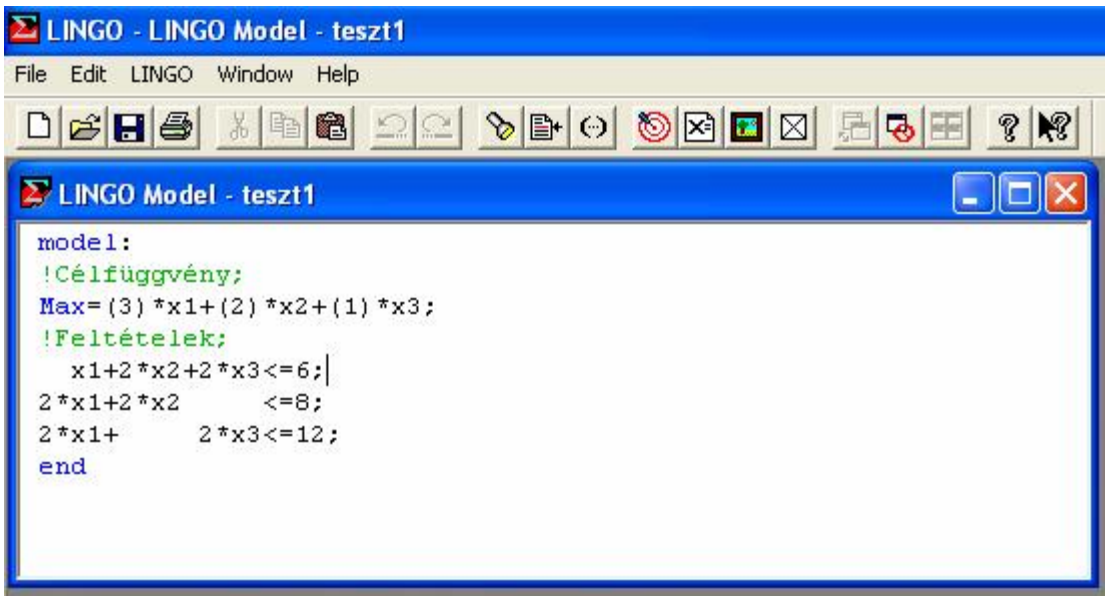
$$x_1 + 2 * x_2 + 2 * x_3 \leq 6$$

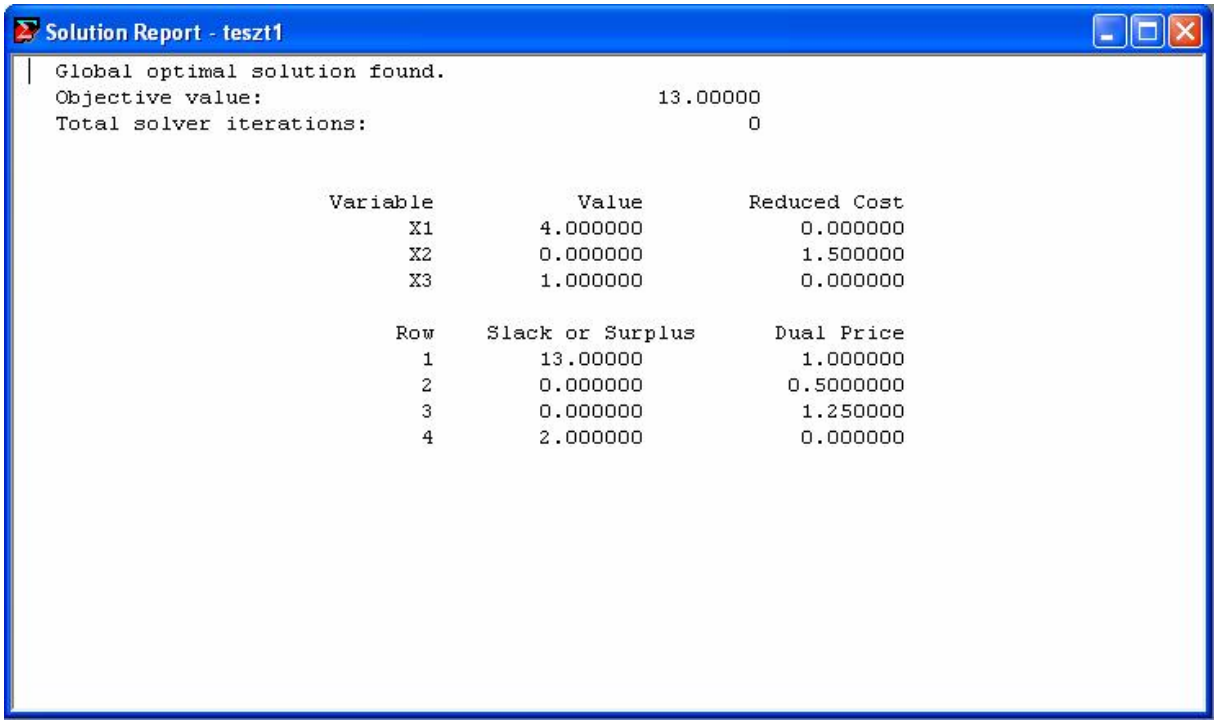
$$2 * x_1 + 2 * x_2 \leq 8$$

$$2 * x_1 + 2 * x_3 \leq 12$$

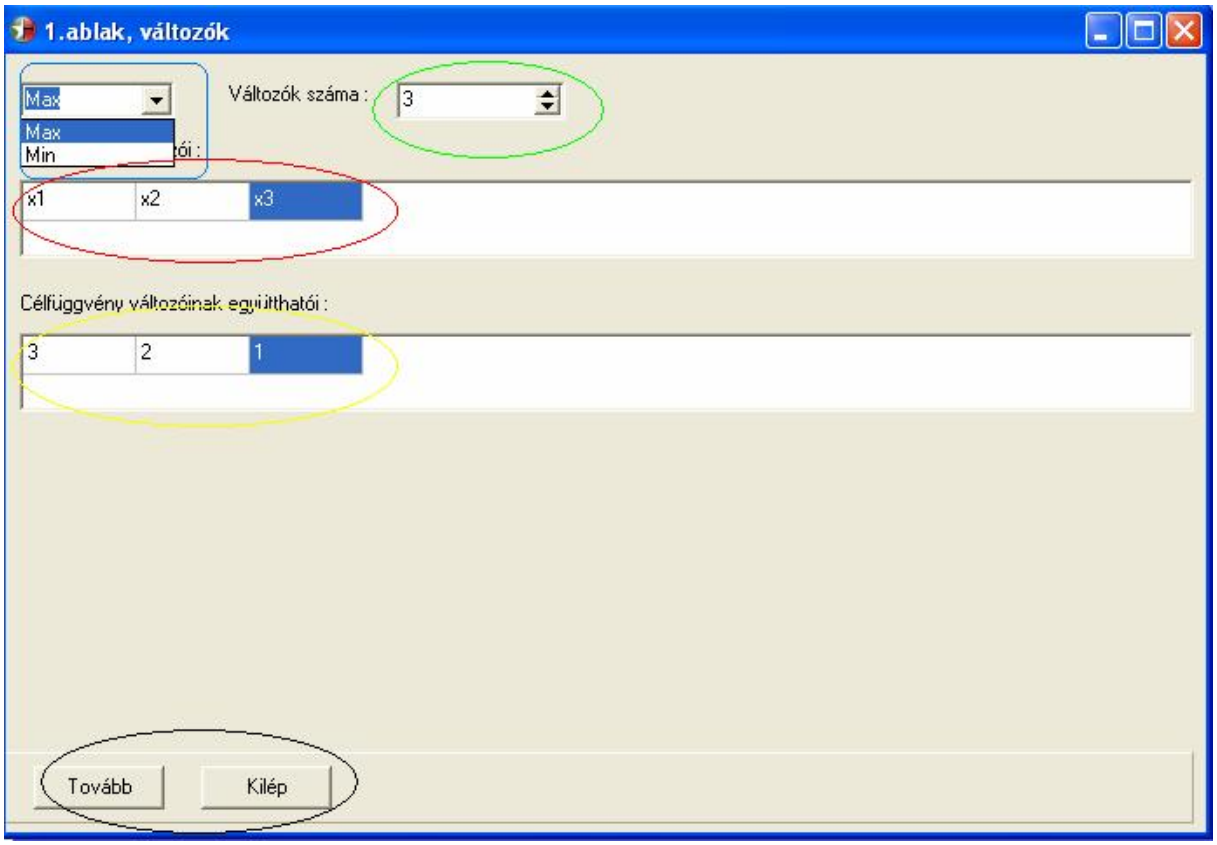
és $x_1, x_2, x_3 \geq 0$.

Oldjuk meg először Lingo10 segítségével, begépeljük a Lingo szintaktikájának megfelelő lineáris programozási feladatot, majd megoldjuk. A következő ablakokat kapjuk, a másodikon az eredmény is kiolvasható, az optimális megoldás: 13.



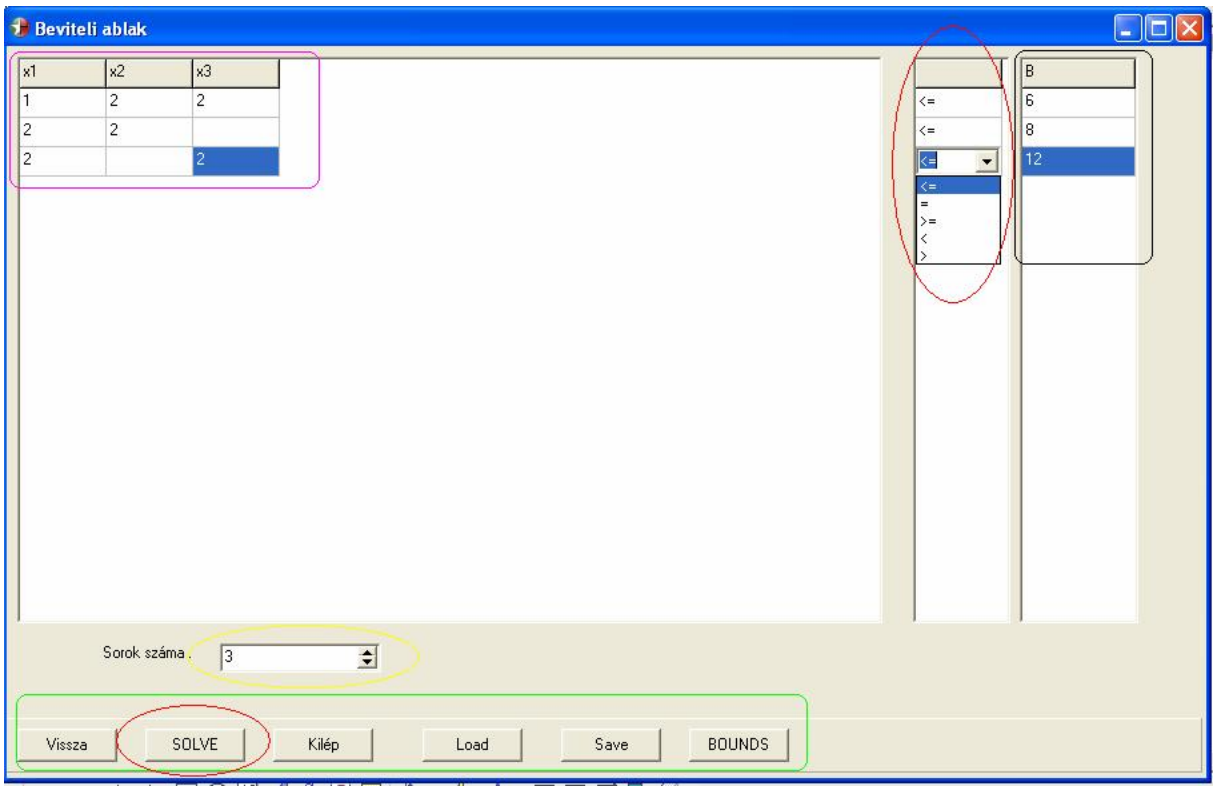


Ezek után oldjuk meg a problémát az általam fejlesztett alkalmazással. A program indulásakor egy információs ablak jelenik meg, amelyet, ha elolvastunk, csak megnyomjuk az OK gombot és máris a célfüggvény megadására szolgáló képernyőn találjuk magunkat.



A fenti képen igyekeztem eltérő színekkel ellátni a különböző funkciócsoportokat. Elsőként a kék színnel keretezett lenyíló listából kell kiválasztani, hogy maximalizálni vagy minimalizálni szeretnénk. A zölddel bekarikázott SpinEditor segítségével kiválaszthatjuk, hogy hány darab változónk van, a példánkban 3 darab szerepel (x_1,x_2,x_3). Ezek után a pirossal és sárgával karikázott részen már csak 3-3 mező fog megjelenni. A pirossal jelölt részben adhatjuk meg a változóink neveit, például x_1 vagy YTWO vagy alma. A sárgával kijelölt részen adjuk meg a változóink együtthatóit. Azokat az együtthatókat, amelyekkel a célfüggvényben szerepelnek!

Ha ezeket az adatokat megadtuk, akkor a „Tovább” gombbal navigálhatunk a feltételek megadásához szükséges ablakhoz vagy kiléphetünk.



A feltételeket a fent látható képen bekarikázott funkciók alkalmazásával adhatjuk meg. Először sárgával jelölt SpinEdit fel-le nyomkodásával állíthatjuk be, hogy hány darab feltételünk lesz (példánkban 3 darab van). A lilával megjelölt szekcióban adhatjuk meg a lineáris programozási feladat „A” mátrixát, azaz a feltételekben szereplő változók

együtthatóit. A változók neveit tartalmazó sort nem módosíthatjuk, azt az előző oldal beállításai alapján automatikusan tölti ki a program. Ha valamelyik cellába nem írunk értéket, akkor oda feldolgozás során 0-t írunk automatikusan. A cellában szereplő értékeket, ha negatívak nyugodtan ’-’ jellel írhatjuk, például -4. Tizedes törtek esetén akár pontot (.), akár vesszőt (,) írhatunk, ugyanis a Lingo felé küldött .lng fájl már csak pontot fog tartalmazni.

A feketével keretezett részbe lehet beírni a probléma „B” vektorát, a feltételek jobb oldalát. A pirossal bekarikázott szekcióban lehet, lenyíló lista segítségével, kiválasztani a feltételek aktuális sorának relációs jelét.

Zölddel jelöltem ábrámon a gombokat. A „Vissza” gombbal visszatérhetünk az előző ablakhoz és a lineáris programozási feladat célfüggvényét módosíthatjuk.

A „Kilép” gomb lenyomása után kilépünk a programból.

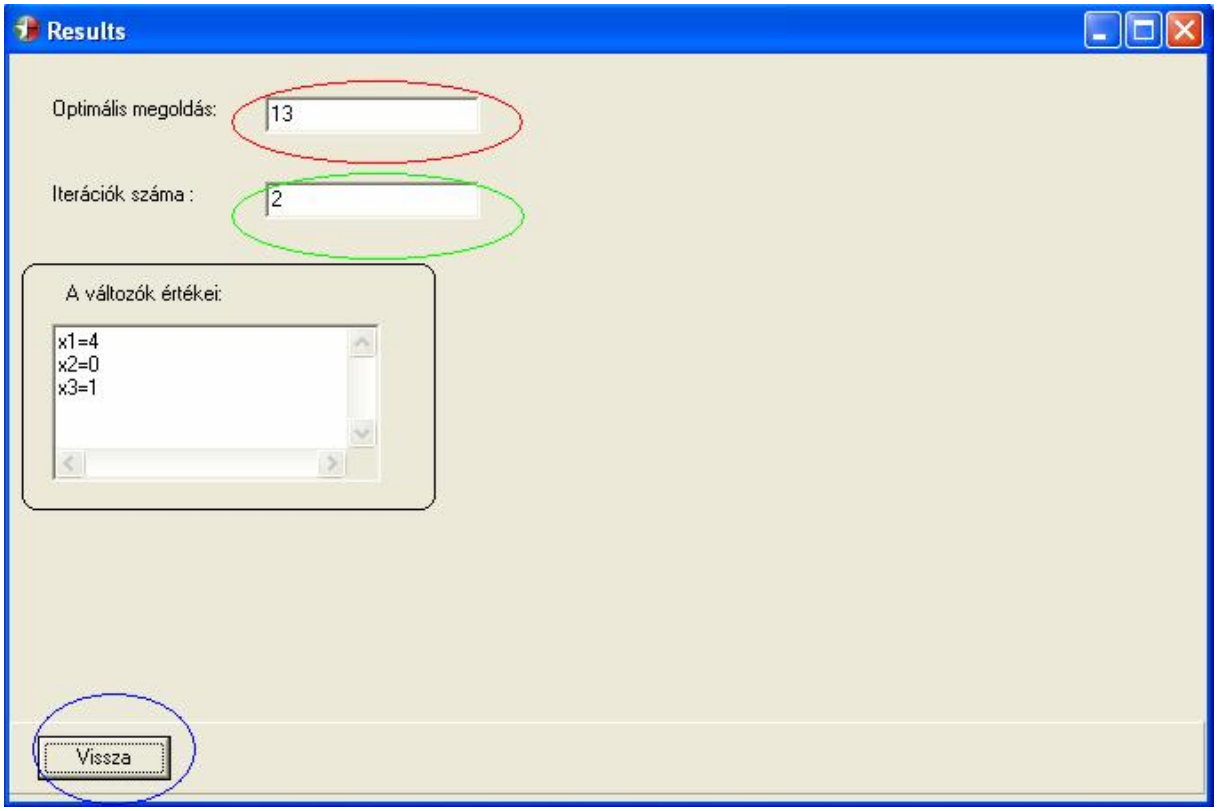
A „Load” gomb lenyomásakor a Windows - ban már megszokott párbeszéd ablak jön fel. Betöltéskor mindig ellenőrizzük le, hogy az 1. ablakon Min vagy Max van kiválasztva!

A „Save” gomb aktiválásakor, természetesen a felhasználó által szerkesztett feladatot lehet lementeni.

A „Bounds” gomb használatáról már írtam egy másik fejezetben, de aktuális példánk nem használ különleges megszorításokat.

Az egyik legfontosabb gomb a „SOLVE” nevet viseli. Ezzel lehet betöltött / begépelt / módosított lineáris programozási feladatot megoldani.

A Solve gomb lenyomásakor beolvassuk a képernyőkről az adatokat, létrehozzuk .lng állományunkat és minden beállítás után átadjuk a vezérlést a Lingo „fekete dobozának”, majd visszakapjuk a megoldást, az előző fejezetben leírtak szerint. Nézzük, hogyan is néz ki egy sikeres és optimális megoldással rendelkező feladat végeredménye.



Ezen a képen már ténylegesen, csak információkat jelenítek meg, itt módosításra nincs lehetőség. Egyetlen gombot tartalmaz az aktuális Form, mégpedig a kékkel bekarikázott „Vissza” gombot, amivel a feltételek beállítására szolgáló képernyőre navigálhatunk.

A megjelenített adatok közül az optimális megoldást pirossal jelöltem, az iterációk számát zölddel karikáztam be, míg a változókat fekete keretben találhatjuk meg az ábrán. Több változó esetén a „A változók értékei” szövegdobozban a görgetősávok aktívvá válnak automatikusan.

9. Összefoglalás

Kutatásom célja az volt, hogy létrehozzak egy olyan alkalmazást, amellyel képes lehet egy egyszerű felhasználó a lineáris programozási feladatot megoldani. A megoldáshoz nem kell ismerni a Lingo (A Lindo operációkutatással foglalkozó vezető fejlesztési központjának terméke) modellezési nyelv szintaktikáját, speciális nyelvi elemeit. Az implementáláshoz Borland Delphi 5 fejlesztőeszközt használtam, amelynek előnyeit és hátrányait is megtapasztalva, sikerült Windows platform alá grafikus környezetet fejlesztenem, amivel képesek vagyunk lineáris programozási feladatokat megoldani.

Az alkalmazásom tartalmaz olyan funkciókat és beviteli segítséget is, amivel akár bonyolultabb operációkutatási feladatokat is meg lehet oldani. Feladatválasztásomat természetesen az is befolyásolta, hogy az egyetemi oktatás során az Operációkutatás tantárgy keretein belül a Lindo programjával a Lingoval ismerkedhettünk meg.
Kutatásomhoz segítséget témavezetőmtől Dr. Bajalinov Erik tudományos főmunkatárs úrtól kaptam, illetve internetes forrásokból. Magyar nyelvű leírást ezekben a témákban alig találtam, általában egyetemi, főiskolai vagy egyéb tanfolyami jegyzetek voltak. Legtöbb jegyzet közgazdászoknak készült és a szimplex-módszert lépéseit mutatja be lépésről-lépésre. Delphi fejlesztési tanácsokat és használható segítségeket lehet találni magyar és idegen nyelven egyaránt.

Az alkalmazásom során nagy hangsúlyt fektettem arra, hogy azt a programot, akik ténylegesen használják majd, egyszerűen lehessen használni. A program gyorsan és helyesen működik. A kutatásom legfontosabb feladata az volt, hogy a Delphi5, mint az Object Pascal objektumorientált nyelve és a Lingo közti kommunikációs csatornát (interfészt) megvalósítsam. A Lindo cég nagyon sok segítséget nyújt, hiszen jól szerkesztett leírást és példa kódrészeket is mellékel, ha letöltjük a próba licences változatot.

Továbbá sikerült megvalósítanom, hogy a felhasználók által begépelt adatokat mielőtt vagy miután megoldottuk, MPS formátumú fájlba lehet elmenteni. Komoly kihívás volt ez számomra, hiszen a célfüggvényben lévő változókra sok feltételt szabhatunk ki, továbbá az értékkészletüket kiterjeszthetjük, illetve szűkíthetjük.

Természetesen nemcsak menteni tudunk MPS formátumú fájlba, hanem azokat beolvasni is képes a programom.

A feladatomat sikerült a kutatásom során megvalósítani, hiszen az applikációm megvalósítja a célul kitűzött funkciókat, a lineáris programozási feladat megoldásához csak a Lingo dll fájlokat használja. A programot bármilyen számítógépen futtatni tudtam, amelyikre felmásoltam Delphi által generált exe fájlomat és a Lingo honlapjáról letöltött dll – t. Bár meg kell jegyeznem, ez sajnos csak MS Windows környezetben működik.

Ha még egyszer nekifognék, vagy javítani szeretném az alkalmazásomat, akkor már beépíteném a programomba azokat a lehetőségeket is, hogy a megoldás előtt ki lehessen választani melyik cég Solver – ét szeretném felhasználni, mert az operációkutatási alkalmazások piacán sok szereplő folyamatosan fejleszt megoldásokat. Ezen cégek, például az LGO vagy LPSolve termékeivel hasonló módon lehetne interfészt megvalósítani. Így egy komolyabb, Solvereket is összehasonlító programot lehetne fejleszteni.

Köszönetnyilvánítás:

Ez úton szeretnék köszönetet mondani a családomnak, barátaimnak, kollégáimnak és nem utolsó sorban témavezetőmnek, Dr. Bajalinov Erik, tudományos főmunkatárs úrnak, diplomadolgozatom elkészítésében nyújtott segítségéért, és tanácsaiért.

Irodalomjegyzék:

Felhasznált irodalom illetve linkgyüjtemény:

MPS formátumról:

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

http://lpsolve.sourceforge.net/5.1/mps-format.htm

http://www.inf.unideb.hu/~bajalinov/lp/4/MPS.pps

Lineáris programozásról angolul és magyarul:

http://en.wikipedia.org/wiki/Linear_programming

https://miau.gau.hu/mediawiki/index.php/Lineáris_programozás

A Lindo cég honlapja:

www.lindo.com

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

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

Operációkutatásról:

https://miau.gau.hu/mediawiki/index.php/Operációkutatás

http://www.hors.hu/rapcsak.htm

Az operáció kutatás és szimplex-módszer „atyjáról”:
http://en.wikipedia.org/wiki/George_Dantzig

Aki talán elsőnek foglalkozott 1939ben operáció kutatással:
http://en.wikipedia.org/wiki/Leonid_Kantorovich

Kuzmina Jekatyerina – Dr. Tamás Péter – Tóth Bertalan : Programozzunk Delphi 7 rendszerben! (Computer Books 2006, ISBN: 963 618 307 4)

Egyéb linkek, ahol Delphi segítséget kaphatunk:
http://www.borland.hu
http://www.prog.hu
http://www.marcocantu.com