*SZOFTVER**

HANÁK GÁBOR

Az MPSX-MIP/370 matematikai programozási programcsomag. Néhány konkrét tapasztalat

Bevezetés

Az MPSX-MIP/370 olyan IBM termék, amelyet a 370-es gépcsaládhoz tartozó számítógépekre lehet installálni. Segítségével vegyes egészértékű lineáris programozási feladatokat oldhatunk meg. Tulajdonképpen két, részben független programcsomagról van szó. Az egyik, az MPSX/370 (*M*athematical *P*rogramming *S*ystem E*x*tended/370) csak folytonos feladatok megoldására alkalmas, viszont a másiktól függetlenül is használhatjuk. A MIP/370 (*M*ixed *I*nteger *P*rogramming) programcsomagot csak az MPSX/370-nel együtt működtethetjük.

Mi két gépen is használtuk az MPSX-MIP/370 programcsomagot. Először a Magyar Tudományos Akadémia Számítástechnikai és Automatizálási Kutató Intézet (MTA SZTAKI) IBM 3031 típusú, majd a Magyar Alumíniumipari Tröszt (MAT) IBM 4331 típusú gépén. Megjegyezzük, hogy információink szerint [5] az ESZR számítógépcsalád nagyobb gépein, például a SZTAKI R-35 típusú gépén is installálható a programcsomag.

Ismertetőnket a következők szerint építjük fel:

1. Általános ismertető az MPSX/370 programcsomagról
2. Általános ismertető a MIP/370 programcsomagról
3. Általános ismertető az MPSX/370 által biztosított Report Generator-ról
4. Konkrét tapasztalatok

1. Általános ismertető az MPSX/370 programcsomagról

A programcsomag alkalmas minden olyan folytonos lineáris programozási feladat megoldására, vizsgálatára, amelyben a feltételek száma nem haladja meg a 16 ezret. (Nem volt alkalmunk kipróbálni ilyen méretű feladat esetén az MPSX/370 működését; saját tapasztalataink alapján azonban elmondhatjuk, ha a feladat ezer körüli feltételt tartalmaz, a programcsomag működése igen jó.) Az egyes feltételek, illetve változók számára bármilyen korlát-intervallum előírható, megengedve, hogy az intervallum végpontjai $\pm\infty$ is lehetnek.

Az MPSX/370-et használhatjuk bizonyos szeparábilis problémák megoldására is, amikor minden nemlineáris függvény egyváltozós konvex nemlineáris függvények lineáris kombinációja.

Az MPSX/370 többféle operációs rendszer alatt is futtatható. A leggyakoribbak a batch futások, ilyenkor vagy OS/VS vagy DOS/VS alatt fut a jobunk.

* Új rovatunk indulásakor a szerkesztő, a *magyangol* nyelv esküdt ellensége, eleve feladta a harcot a számítástechnikusok túlerejével szemben.

Lehetséges azonban a programcsomagot az interaktív CMS (*C*onversational *M*onitoring *S*ystem) alatt is használni. Ez utóbbira természetesen csak akkor van mód, ha a feladat méretei nem túl nagyok. Mindenképpen hasznos viszont a CMS alatti használat részfeladatok, programrészletek tesztelésére. Mivel az MPSX/370 leírása az operációs rendszertől lényegében független, az OS és a DOS futás különbözőségeire csak az utolsó részben térünk ki.

Először az MPSX/370 belső logikájáról szólunk, a konkrét módszerekről később lesz szó.

Egy MPSX/370 job összeállításához job-kontrol utasításokra, kontrol-program utasításokra és esetleg adatokra van szükség. A job-kontrol utasítások annyiban függenek a konkrét megoldandó feladattól, amennyiben ez meghatározza az MPSX/370 által megkövetelt, az adott futáshoz szükséges file-ok számát, neveiket, méreteiket és esetleges egyéb jellemzőiket. A kontrol-programot kétféle program-nyelven írhatjuk meg. Az egyik, az MPSCL (*M*athematical *P*rogramming *S*ystem *C*ontrol *L*anguage) MPSX/370 specifikus, azaz az ilyen nyelven írt kontrol-programot az MPSX/370-be beépített fordító, szerkesztő, futtató program kezeli. Ez a lehetőség nagymértékben egyszerűsíti a kontrol-program megírását. Hátránya viszont, hogy nem teszi lehetővé a nehezebb problémák megoldását. Erre alkalmas az ECL (*E*xtended *C*ontrol *L*anguage) nyelv, amely tulajdonképpen a PL/I nyelv bővítésének tekinthető. Egy ECL nyelven megírt kontrol-program olyan PL/I program, amely speciális MPSX/370 makró- illetve eljárás-hívásokat is tartalmaz. Az ECL nyelv használatakor a kontrol-programot a PLIOPT fordítóprogrammal fordítjuk. Kontrol-programunkban tehát minden olyan PL/I utasítást használhatunk, amely a PLIOPT számára értelmezhető.

Ebben az ismertetőben csak az ECL nyelven írt programokkal foglalkozunk. Egy ilyen program első utasítása:

címke: PROCEDURE OPTIONS (MAIN);

második utasítása:

%INCLUDE DPLINIT;

utolsó utasítása:

END címke;

kell legyen. A második utasítással aktivizáljuk a DPLINIT nevű MPSX/370 makrót. Ez részben standard induló értékeket ad a program futását befolyásoló különböző paramétereknek, részben pedig arról intézkedik, mi történjék, ha bizonyos események bekövetkeznek.

Az MPSX/370 paramétereit ún. kommunikációs cellákban tárolja. Amennyiben a standard implementációtól eltérő paraméter értékeket szeretnénk használni, úgy a DPLINIT makró meghívása, azaz a kezdőértékek beállítása után a kontrol-programban újradefiniálhatjuk a kívánt kommunikációs cellák tartalmát: ily módon választhatunk a programcsomag által kínált lehetőségek közül.

Egy MPSX/370 futás során különböző fontosabb „események" következhetnek be, például az első feasibilis megoldás elérése, optimális megoldás elérése, kiderül, hogy nincs feasibilis megoldás, vagy van, de nem korlátos, újra kell invertálni a mátrixot, ki kell nyomtatni az aktuális megoldást, különböző hibák,

stb. Ezekre az „eseményekre" a DPLINIT standard választ ad. Ha például ún. fő-hiba következik be, akkor, a standard implementáció szerint, a futás a STATUS eljárás (ld. később) meghívása után befejeződik. A különböző „eseményekre" adott választ végső soron természetesen a felhasználó határozza meg, hiszen a DPLINIT meghívása után felülírhatja a standard választ. Ha például egy konkrét feladat esetében tudjuk, hogy a standard implementáció szerint rengeteg nyomtatás várható, akkor az XDOPRINT választ felülírjuk és kihagyjuk belőle a nyomtatási utasítást; ez esetben viszont a nyomtatásról magunknak kell gondoskodnunk a kontrol-programban. Más példa: bizonyos, előre sejthető hiba előfordulása esetén ne álljon le a futás, hanem valamilyen MPSX/370 eljárás segítségével próbáljuk meg rögtön felderíteni, esetleg elhárítani a hibát. Ez utóbbi esetben persze nagyon óvatosan kell eljárnunk, nehogy a program végtelen ciklusba essék. (A különböző „események" és a rájuk adott standard válaszok felsorolása az [1] Demands fejezetében található meg.)

A legegyszerűbb ECL programok a DPLINIT makró meghívása után lényegében kétféle utasítást tartalmaznak: értékadást és eljárás hívást. Az értékadással valamilyen kommunikációs cellát töltünk meg adattal, illetve módosítjuk annak értékét. Az eljárás hívás egy CALL utasítás, amely valamilyen speciális MPSX/370 vagy MIP/370 eljárást aktivizál.

Az MPSX/370 eljárásokat hét csoportra bontva tárgyaljuk. Ezek:

1.1 Problem file fenntartási eljárások
1.2 Induló eljárások
1.3 Optimalizáló eljárások
1.4 Eredmény nyomtatási eljárások
1.5 Paraméteres eljárások
1.6 Mentési és újraindítási eljárások
1.7 File kezelési eljárások

Nem célunk a mindent átfogó ismertetés, ezért csak a fontosabb eljárásokról szólunk.

1.1 *Problem file fenntartási eljárások*

Logikai szempontból az első ilyen eljárás a *CONVERT*: a felhasználó által valamilyen formában (ld. erről később) megadott adatrendszert bináris pakolt formátumúvá konvertálja, és egy, szintén a felhasználó által adott név hozzárendelésével elhelyezi a problem file-on, amennyiben ez utóbbi egyáltalán létezett; ha nem, akkor előbb magát a problem file-t is elkészíti.

A problem file alapvető jelentőségű MPSX/370 objektum. Itt lehet tárolni — az MPSX/370 számára kezelhető formában — feladatunk adatrendszerét (egy konkrét feladathoz akárhány jobb oldalt, ún. range, ill. bound vektort rendelhetünk); itt őrződnek meg az egyes futások alkalmával előálló, az adott problémához tartozó, névvel ellátott bázisok (pl. ugyanannak a feladatnak más jobb oldallal, más célfüggvénnyel való futása esetén célszerű alkalmas nevek használata), valamint vegyes feladat esetében az integer fázisban előállt különböző, szintén névvel ellátott fák (erről ld. később). Egy problem file-on tetszés szerinti számú egymástól független probléma tárolható a hozzá tartozó adat-, bázis- ill. fa-rendszerrel együtt.

A CONVERT bizonyos szintű automatikus hibakeresést végez, ezenkívül a konvertált problémáról statisztikát készít. A megfelelő paraméterek beállításával mind a hibakeresés, mind a statisztika alaposabbá tehető.

A problem file-on létező valamilyen problémát kétféle eljárással változtathatjuk meg. Az egyik a *REVISE*, ennek hatására új probléma keletkezik a problem file-on. A REVISE segítségével vektorokat törölhetünk az eredeti problémából és új vektorokat adhatunk hozzá; bármely adatát módosíthatjuk. Ha az új problémának ugyanazt a nevet adjuk, mint az eredetinek, úgy a REVISE a régit felül fogja írni. Van olyan eset, amikor célszerű ily módon eljárni (például helytakarékossági okok miatt), de általában biztonságosabb az új problémának új nevet is adni: ez esetben mindkét probléma megőrződik a problem file-on.

A másik eljárás, amivel a problem file-on adott valamilyen problémát megváltoztathatunk a *MODIFY*. Ez az eljárás azonban nem a problem file-on operál, hanem az ún. work matrix-on. Az 1.2 pontban tárgyalandó SETUP eljárás a problem file-on adott probléma alapján elkészíti a work matrix nevű file-t. A MODIFY ezen a mátrixon hajt végre változtatásokat. Vektorokat nem lehet törölni, ill. hozzáadni, csak a meglevő adatokat módosíthatjuk.

Mikor célszerű a REVISE-t, és mikor a MODIFY-t használni? Ha korlátlanul áll rendelkezésünkre szabad lemezterület, akkor, a problem file-t elég nagynak meghatározva, célszerű a REVISE-t használni. Általában azonban nem ez a helyzet. Ezért a következő eljárást javasoljuk: egy adott probléma nagyobb átalakítása esetén használjuk a REVISE-t; ha viszont csak pár adatot érintő változatásról, esetleg ilyen változtatások seregéről van szó, használjuk a MODIFY-t. Ez utóbbi esetben a keletkező optimális megoldásnak, ill. fának könnyen felismerhető nevet adunk; ez a bázis, ill. fa a megadott névvel felíródik a problem file-ra.

A REVISE használata esetén például problem file-unkon a következők találhatók: *PROB1*, bázis, fa; *PROB2*, bázis, fa; . . . ; *PROB12*, bázis, fa. A másik esetben pedig: *PROB1*, bázis ALAP, fa ALAP, bázis KOZEP, fa KOZEP bázis NAGY, fa NAGY, bázis ORIAS, fa ORIAS; *PROB2*, bázis ALAP, fa ALAP, bázis KOZEP, fa KOZEP, bázis NAGY, fa NAGY, bázis ORIAS,, fa ORIAS; *PROB3*, bázis ALAP, fa ALAP, bázis KOZEP, fa KOZEP, bázis NAGY, fa NAGY, bázis ORIAS, fa ORIAS.

Természetesen a REVISE használata esetén az új problémához csatolva megjelennek a régi problémához tartozó bázisok, ill. fák eredeti nevükön, de a feladat módosításának megfelelően módosítva. Ily módon természetesen előfordulhat, hogy egy a régi problémához tartozó optimális bázis a módosítás után nem lesz az, mégis érdemes az új probléma optimális megoldásának keresésekor ezt a bázist használni induló bázisként, mert ily módon nagyon hamar eljutunk az új problémánál érvényes optimumhoz.

A problem file tartalomjegyzékét a *PROBLEMS* eljárással kérdezhetjük le. Ugyancsak ezzel az eljárással lehet egy adott problémát, illetve a hozzárendelt bázist és fát törölni vagy átnevezni.

1.2 *Induló eljárások*

A legfontosabb induló eljárás a *SETUP*, amely hármas funkciót lát el. Először is kijelöli a feladatnak adekvát helyigényt és inicializál bizonyos későbbi eljárások által használt file-okat. A helyigény kijelölését három független paraméterrel irányíthatjuk, de egyszerűbb esetekben célszerű ezek alapértelmezésére hagyatkozni; tényleges használatukra csak problematikus esetben, az MPSX/370 alapos ismeretében kerülhet sor. Másodszor, a SETUP létrehozza a work matrix-ot, figyelembe véve a feladat összes változóját, feltételét és jobb

oldalát, de csak egy range és egy bound vektort engedve meg (a range a feltételekre, a bound a változókra ad korlátot). Végül a SETUP meghatároz egy kezdő megoldást is.

Az MPSX/370 minden egyes feltételhez hozzárendel egy ún. logikai változót. Ily módon egy feladathoz annyi MPSX/370 változó tartozik, mint az eredeti változók és feltételek számának összege. Az eredeti változókat strukturális változóknak nevezzük. A logikai változókat mint az adott feltételhez tartozó slack-változókat fogjuk fel.

A SETUP által meghatározott induló megoldásban a bázist a logikai változók alkotják, míg a strukturális változók az alsó ill. felső korláttal indulnak. Ezt az induló helyzetet all-slack állapotnak nevezzük.

A SETUP eljárás során kell intézkednünk arról, hogy a későbbi optimalzálás során minimalizálni vagy maximalizálni akarunk-e. Az alapértelmezés a minimalizálás.

A problem file-on adott problémának többféle redukcióját végezhetjük el itt nem részletezendő eljárások segítségével. Ezek részben kiszűrik a feladatban levő redundanciákat és a redukált feladatra vonatkozóan végzik el a SETUP funkcióit. Másrészt a SETUP végrehajtása után módosíthatjuk a generált work matrix-ot (például a fentebb ismertetett MODIFY segítségével). Természetesen az ilyen változtatások nem érintik a problem file-on levő adott problémát.

1.3 *Optimalizáló eljárások*

Az MPSX/370 programcsomagban többféle optimalizáló eljárás közül választhatunk. A legalapvetőbb ilyen a PRIMAL. Nagyobb feladatok esetén célszerű a PRIMAL-t előkészíteni a CRASH-sel.

A *CRASH* eljárás segítségével részben igyekszünk minél több strukturális változót bevonni a bázisba, részben az infeasibilitás nagyságát próbáljuk csökkenteni. A CRASH két lépésből áll. Az elsőben választhatunk: vagy a strukturális változók bázisba való bevonása (ez az alapértelmezés), vagy pedig az infeasibilitások négyzetösszegeinek redukciója az elsődleges cél. A második lépésben a CRASH az infeasibilitások abszolút értékei összegének csökkentésére törekszik.

Ha az aktuális bázis már nem all-slack állapotú (pl. a CRASH után, vagy ha egy korábban mentett feladatot indítunk újra), az optimális megoldást legcélszerűbb a *PRIMAL* eljárással keresni. A PRIMAL a kétfázisú módosított szimplex módszert használja.

Az optimalizálás menetét több paraméterrel kontrollálhatjuk, de lehetőség van a DPLINIT által beállított alapértelmezések használatára is. Ilyenkor csak annyi a dolgunk, hogy bármilyen paraméter megadása nélkül meghívjuk a PRIMAL eljárást.

A PRIMAL standard meghívása dinamikus pricing és cycling módszert használ annak eldöntésére, hogy a soron következő iteráció mely vektort vigye ki a bázisból, és melyiket hozza be.

Egy másik optimalizáló eljárás a *DUAL*, amely azonban elsősorban csak megengedett megoldás elérésére törekszik. Létjogosultsága azért van mégis, mert (a PRIMAL-tól eltérően) az eljárás során a feasibilitások csökkentése mellett a célfüggvény javítása is cél. A DUAL eljárás végén egy megengedett megoldás áll rendelkezésünkre (hacsak létezik ilyen), amely ugyan gyakran nem optimális, viszont az optimumhoz sokkal közelebb áll, mint a PRIMAL első

megengedett megoldása. Természetesen ha optimális megoldást akarunk, úgy a DUAL után meg kell hívnunk a PRIMAL-t is.

Végül az *OPTIMIZE* makró segítségével automatikusan meghívhatjuk a fentebb ismertetett optimalizáló eljárásokat a programcsomag által meghatározott módon. Természetesen itt is van mód bizonyos paraméterek beállításával befolyásolni a futás menetét, de — főleg az MPSX/370 legegyszerűbb használata esetén — elég pusztán az OPTIMIZE makrót meghívni, és akár all-slack állapotból kiindulva optimális megoldáshoz juthatunk.

1.4 *Eredmény nyomtatási eljárások*

A legalapvetőbb eljárás, amellyel információkat szerezhetünk az aktuális megoldásról, a *SOLUTION*. A SOLUTION meghívásával a következő nyomtatott outputhoz juthatunk:

— általános jellemzők (nem megengedett, megengedett ill. optimális-e a megoldás; iterációszám; célfüggvény értéke stb.)

— az egyes feltételekről: sorszáma; neve; benne van-e a bázisban, vagy alsó ill. felső korláton van; sorösszeg; slack; alsó korlát; felső korlát; duális aktivitás

— az egyes változókról: sorszáma; neve; benne van-e a bázisban, vagy alsó, ill. felső korláton van; aktuális érték; input cost; alsó korlát; felső korlát; reduced cost.

A lista jelzi azt is, hogy — nem megengedett megoldás esetén — hol van infeasibilitás, ill. — optimális megoldás esetén — hol van alternatív optimum. Van lehetőség a fent felsorolt információk részleges lekérdezésére is.

Az optimalitás elérése után a megoldás érzékenységére nézve kaphatunk információkat a *RANGE* eljárás segítségével.

Kinyomtathatjuk a szimplex táblát oszloponként *(TRANCOL)* vagy soronként *(TRANROW)*.

Infeasibilitás esetén meghívhatjuk a *TRACE* eljárást, amely segít az ok felderítésében.

Kinyomtathatjuk az adott feladathoz tartozó bemenő adatrendszert. Kártyakép formában megjeleníthetjük az adatokat a *BCDOUT* eljárás segítségével, amely oszlopfolytonos listát nyújt. (Sorfolytonos listát a fentebbi TRANROW eljárással kaphatunk.) A *PICTURE* eljárás viszonylag jól áttekinthetően, mátrix-szerűen közli az adatokat, pusztán azok nagyságrendjének jelzésével.

Az ECL nyelv lehetőséget ad arra, hogy ne a nyomtatóra vagy egy külső file-ba küldjük, hanem egy ún. ECL-struktúrába töltsük be az adatokat. Az ECL-struktúra nem más, mint az ECL nyelv által használt speciálisan felépített PL/I struktúra. Az *INQUIRE* eljárás segítségével valósíthatjuk meg az adatok ECL-struktúrába töltését. Ily módon rekurzív feladatok igen könnyen kezelhetővé válnak.

Végül a *STATUS* eljárás segítségével információkat kaphatunk a gép, a megoldás, a mátrix és a toleranciák aktuális állapotáról.

1.5 *Paraméteres eljárások*

Az MPSX/370 ötféle paraméteres eljárása a következő elven alapszik: adott egy vektor, ezt szeretnénk paraméterezni; adott egy másik vektor, a paramétervektor; végül adott egy szám, ami a paraméter felső korlátja lesz. Az eljárás során a paraméterezendő vektor helyett a következő vektor kerül a work mat-

rixba: az eredeti vektor plusz a paramétervektor és a paraméter szorzata. A paraméter nullától folytonosan nő egészen a felső korlátig. Közben az eljárás báziscserékkel tartja fenn az optimalitást (feasibilitást). Az eljárás során minden báziscserénél kinyomtatásra kerül az iterációszám, a bázisból kimenő ill. oda bemenő vektor sorszáma, a célfüggvényérték és a paraméterérték. Az eljárást bármelyik iterációnál ill. a paraméter bizonyos értékeinél félbeszakíthatjuk. Ilyenkor meghívhatunk bármilyen nyomtatási eljárást (ld. 1.4 pont), majd a paraméteres eljárás folytatódhat.

Az MPSX/370 programcsomagban a következő paraméteres eljárások vannak beépítve:

— *PARAOBJ*, paraméterezendő a célfüggvény vektor
— *PARARHS*, paraméterezendő a jobb oldal
— *PARARIM*, paraméterezendő a célfüggvény és a jobb oldal egyszerre
— *PARACOL*, paraméterezendő az egyik strukturális változó
— *PARAROW*, paraméterezendő az egyik feltétel vektor.

Természetesen a kontrol-programból a feladat bármilyen paraméterezését megvalósíthatjuk, különösen alkalmas erre az ECL nyelv.

1.6 *Mentési és újraindítási eljárások*

Négy ilyen eljárás van az MPSX/370 programcsomagban. Két eljárás mentő, két eljárás újraindító. Másrészt két eljárás, egy mentő és egy újraindító, a problem file-lal kommunikál, míg a másik kettő a rendszer perifériáival, ill. ECL-struktúrákkal.

Az aktuális bázist menti a problem file-ra a *SAVE* eljárás. Egy adott problémához mentett bázisok sorozata tartozhat. Az egyes bázisoknak nevet lehet adni. A mentett bázist a problem file-ról a *RESTORE* eljárással lehet behívni, természetesen megadva azt a nevet, amelyet az előző SAVE során használtunk.

A *PUNCH* eljárás segítségével az aktuális bázist átadhatjuk a rendszer printernek vagy punchnak, átvihetjük valamilyen mágneses adathordozóra, vagy ECL-struktúrába tölthetjük. Ennek az eljárásnak a megfordítottja az *INSERT*.

1.7 *File kezelési eljárások*

Az *ASSIGN* eljárás arra szolgál, hogy azonosítsuk a kontrol-programban használt MPSX/370 file neveket a job-kontrol utasításokban szereplő adatnevekkel. Természetesen a legtöbbet használt file-ok esetében ez az azonosítás automatikus.

A lineáris programozási feladatok általában megkívánnak bizonyos permanens adatállományokat, azaz olyan file-okat, amelyek nem szűnnek meg egy-egy job lefutása után sem. Mindenekelőtt ilyen a problem file. Másik ilyen a report készítéséhez szükséges file (erről bővebben később). Mindkét előbb említett file-nak közvetlen elérésűnek kell lennie. Ha viszont nincs lehetőség permanens közvetlen hozzáférésű állomány létrehozására, úgy a szükséges file-okat, például a problem file-t, minden egyes job esetében először szalagról egy temporális területre be kell olvasni, majd a job végén azokat szalagra vissza kell tölteni. Ezt a célt szolgálja az *IMPORT* és az *EXPORT* eljárás. Ezek akkor is hasznosíthatók, amikor egy adott problémát fizikailag át kell helyezni egy másik számítógépre.

Az MPSX/370 I/O adatrendszere

Az MPSX/370 számos eljárása input adatokkal dolgozik. Ilyen eljárás például a CONVERT, a REVISE, a MODIFY stb. Az input adatokat többféleképpen is megadhatjuk.

A legalapvetőbb módszer, hogy valamilyen periférián keresztül létrehozunk egy kártyaképekből álló adatfile-t. Ezt Data Deck-nek hívjuk. Ezt a Data Deck-et aztán a job futása során valamilyen perifériáról beolvassuk az MPSX/370 számára. A kártyaképeken az adatok megadása kétféleképpen történhet: kötött ill. szabad formátum szerint, de mindenképpen oszlopfolytonosan: azaz egy változónak az összes feltételben szereplő együtthatóit a Data Deck-nek egymás utáni kártyaképei kell hogy tartalmazzák.

Az ECL nyelv használata viszont lehetőséget ad arra, hogy az input adatokat ne Data Deck-en, hanem ECL-struktúra formában adjuk meg.

Az MPSX/370 által adott output vagy valamilyen perifériára kerül (rendszer printer, punch, lemez, szalag), vagy pedig ECL-struktúrába töltődik. Számos olyan eljárás van (SOLUTION, RANGE stb.), amely ún. standard formátumú file-t tud készíteni (ha a megfelelő paraméterrel lemezre vagy szalagra irányítjuk az outputját); ez olyankor hasznos, ha a kapott eredményeket valamilyen magasszintű nyelven (FORTRAN, PL/I, COBOL) írt programmal további feldolgozásnak vetjük alá.

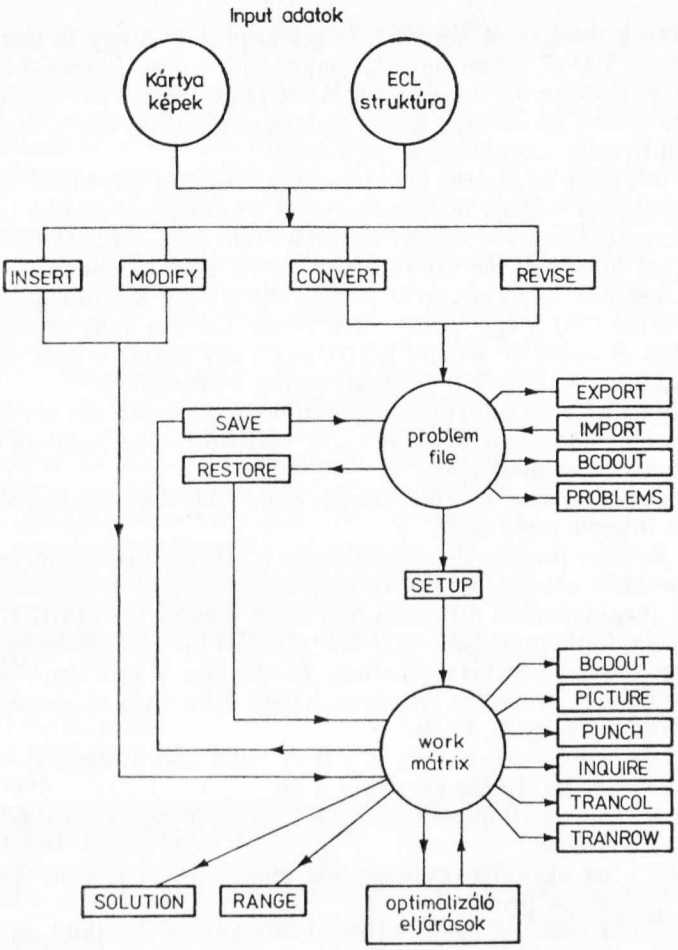
A legegyszerűbb ECL nyelven megírt kontrol-program, amely egy, a rendszer inputon keresztül érkező Data Decken megadott problémát állít fel, old meg, és a megoldás outputját a rendszer printerre küldi, így nézhet ki:

```
PROGRAM: PROCEDURE OPTIONS (MAIN);
%INCLUDE DPLINIT;
    XDATA       = 'ADAT';
    XPBNAME     = 'PBFILE';
    XOBJ        = 'CELFUGGV';
    XRHS        = 'JOBOLDAL';
      CALL CONVERT;
      CALL SETUP;
      CALL OPTIMIZE;
      CALL SOLUTION;
STOP;
END PROGRAM;
```

Az MPSX/370 programcsomag fentebb ismertetett fontosabb eljárásait az 1. ábrán szemléltethetjük:

2. Általános ismertető a MIP/370 programcsomagról

A MIP/370 programcsomag olyan lineáris programozási feladatok megoldására alkalmas, melyben a változók egy részére (vagy egészére) kikötjük, hogy azok értéke a megoldásban csak egész szám lehet. A MIP/370, mint azt a Bevezetésben említettük, nem önálló programcsomag, hiszen csak az MPSX/370-nel együtt működtethető. Egy vegyes vagy tiszta egész feladat esetén először az MPSX/370 programcsomag megoldja a problémát úgy, mintha az szabályos folytonos feladat volna. Azaz minden kikötés ugyanúgy él, mint az eredeti fel-



1. ábra

adatnál, csak az nem, hogy a megfelelő változóknak egészeknek kell lenniük. Az így nyert feladatot nevezzük az eredetihez tartozó folytonos feladatnak. A tulajdonképpeni Mixed Integer fázis, azaz a MIP/370 bekapcsolódása a feladat megoldásába akkor kezdődik, amikor a folytonos feladat optimális megoldását már megkaptuk.

A MIP/370 által irányított futás ezután úgy folytatódik, hogy a programcsomag, a paraméterek beállításától és a feladat természetétől függően, választ egy ún. elágazó változót, amelynek egésznek kellene lennie, de a folytonos optimumban az értéke nem az. A megoldás ezután két ágon fut tovább. Az egyik ágon a kiválasztott elágazó változó felső korlátjának választjuk a folytonos optimum során felvett értékének az egészrészét, míg a másik ágon ugyanezen változó alsó korlátjaként az előző számnál eggyel nagyobbat választunk. Az így kapott két folytonos lineáris programozási feladatot (amelyek egymástól csak az elágazó változóra adott egyedi korlátban különböznek) megoldjuk. Ha ezt az eljárást folytatjuk, elágazási pontok, ill. elágazások, és a hozzájuk tartozó foly-

tonos feladatok rendszerét kapjuk. Az így kapott gráf egy fa lesz. Mivel alapvető kikötés a MIP/370-ben, hogy az egész-változókra kötelező explicit véges alsó és felső korlátot adni, a fának véges sok pontja lesz. Ezért ha létezik az eredeti (vegyes egész) feladatnak megengedett megoldása, úgy a fent ismertetett eljárással optimális megoldáshoz juthatunk.

Úgy tűnhet, hogy az eljárás igen hosszadalmas és a folytonos részproblémák száma az elágazási szintek növekedésével a 2 hatványai szerint nő. Valójában nem ez a helyzet. Bizonyos ágakat ugyanis rövid számolás után azonnal törölni lehet (például infeasibilitás miatt, vagy azért, mert az elérhető célfüggvényérték csak rosszabb lehet egy már megkapott integer megoldáshoz tartozó célfüggvényértéknél stb.) Így aztán nem olyan fát kapunk, amelynek minden pontjából két él indul ki, hanem kettő, egy vagy nulla. A gráf pontjait node-oknak nevezzük. Végül is a következő esetek lehetségesek:

1. a keletkezett gráf nem tartalmaz olyan node-ot, amelyhez tartozó folytonos probléma megoldásában minden egész változó értéke valóban egész; ekkor a feladatnak nincs megoldása.
2. az előző ellentéte, azaz van ún. integer node; ebben az esetben a fa tartalmaz optimális integer node-ot is.

A Mixed Integer problémák megoldására a MIP/370 programcsomag háromféle megközelítési szintet ajánl. Az alapszint lényegében egyetlen makró, az *OPTIMIX* meghívásából áll. Ez a folytonos feladat OPTIMIZE makrójához hasonlóan tulajdonképpen több MIP/370 eljárást hív meg, és az egyszerűbb feladatok megoldására kiválóan alkalmas is. Amikor a probléma bonyolultabb, vagy az OPTIMIX makróba beépített lehetőségek nem elégségesek, a középszint eljárásait hívhatjuk. Ezek:

MIXSTART: előkészítő eljárás a mixed fázis számításaihoz
MIXFLOW: ez az eljárás készíti el a fát
MIXSTATS: statisztikákat közöl a MIXFLOW fázisról és az egész változókról
MIXSAVE: az aktuális problémához rendelve a megadott néven menti a fát a problem file-ra
MIXFIX: rögzíti az egész változók értékét az aktuális egész értékeken további olyan (folytonos) vizsgálatok számára, amikor ezeknek a változóknak az értékét fixen akarjuk tartani és csak arra vagyunk kíváncsiak, hogy a többi változó hogyan viselkedik.

Az OPTIMIX makró tulajdonképpen ennek az öt eljárásnak egymás utáni meghívása standard paraméterekkel.

A Mixed Integer probléma magasszintű megközelítése feltételezi a mixed fázist befolyásoló és egymással igen bonyolult összefüggésben levő összes paraméter hatásának ismeretét. Ismertetésükre itt nem térünk ki.

A legegyszerűbb Mixed Integer problémák esetében tehát az előző fejezet végén leírt ECL kontrol-programon csupán annyit kell változtatnunk, hogy az OPTIMIZE és a SOLUTION között az OPTIMIX-et is meg kell hívnunk:

```
      .
      .
      .
CALL OPTIMIZE;
CALL OPTIMIX;
CALL SOLUTION;
      .
      .
      .
```

3. Report Generator

A most ismertetendő Report Generator nem önálló programcsomag, hanem az MPSX/370 része. Segítségével — az aktuális megoldás alapján — különféle eredményközlő táblázatokat lehet szerkeszteni.

Külön programcsomagként elérhető egy MGRW (*M*atrix *G*enerator and *R*eport *W*riter) nevű önálló IBM termék, amely, a leírás szerint, mind a Data Deck generálásához, mind reportok készítéséhez használható; ráadásul az itt ismertetendő Report Generator lehetőségeinél rugalmasabban és hatékonyabban. Minthogy azonban ezt a programcsomagot csak papírról ismerjük, használni nem állt módunkban, ezenkívül ezt a lehetőséget az MPSX-MIP/370 installációja nem tartalmazza, ismertetésétől eltekintünk.

A Report Generator használata tulajdonképpen egy speciális (a Report Generator nyelvén megírt) program futtatását jelenti bizonyos MPSX/370 eljárások segítségével. A programnyelv a következő típusú utasításokat ismeri:

— numerikus- és karakter-konstansok definiálása
— numerikus- és karakter-konstansok beolvasása
— kiírás a rendszer outputra (printer vagy punch)
— kiírás felhasználói file-ra
— az előző két irány közötti kapcsoló átállítása
— a kiíró utasítás számára számolás
— nem kiírás számára számolás
— tesztelés
— címzés
— makró definiálása
— makró hívása
— comment.

A programnyelv a következő elemeket tartalmazza:

— numerikus- és karakter-konstansok
— numerikus- és karakter-változók
— megoldás-elemek (változók, feltételek alsó- és felső korlátai, azok értékei az aktuális megoldásban, input cost, slack, reduced cost stb.)
— mátrix-elemek
— függvények (ezek a következők lehetnek: abszolút érték, egészrész, az argumentumok minimuma, maximuma és összege, négyzetgyök)

A Report Generatort elsősorban akkor célszerű használni, ha egy adott lineáris programozási feladatot sok szempontból akarunk vizsgálni és sokszor futtatni kis változtatásokkal. Ilyenkor gyakran fölösleges a SOLUTION eléggé „bőbeszédű" outputját böngészni, amely ráadásul sok fontos információt nem mutat ki. A Report Generator lehetőséget nyújt arra, hogy minden futásról ugyanazokat a lényeges információkat kérjük le és így a megoldás jellemzőinek olyan sűrítményét tanulmányozhatjuk, amely „ránézéssel" is jól kezelhető. Nagy hibája a Report Generatornak, hogy egy futás során mindig csak az éppen aktuális megoldást veszi figyelembe, ezért a program-nyelven belül nincs lehetőség két különböző megoldás elemeinek összehasonlítására.

A Report Generator három MPSX/370 eljárást tartalmaz.

Az *ANALYZE* tulajdonképpen fordítóprogram. Inputja vagy a programnyelven megírt program, vagy az ugyanezen a nyelven leírt adatrendszer. (Gyakran szükség van ugyanis olyan adatok bevitelére, amelyek magába a programba — variábilitásuk miatt — nem vihetők be, és sem az aktuális meg-

oldásból, sem a mátrixból nem olvashatók ki). A programot és az adatrendszert az ANALYZE külön kezeli, ezért például ha programot és adatrendszert is akarunk használni, az ANALYZE-t kétszer kell meghívni: egyszer a program miatt, egyszer az adatrendszer miatt. A lefordított programot (adatrendszert) az ANALYZE egy speciális file-on, a REPFILE-on (REPDATA-n) helyezi el. Ez a fordítás még független a konkrét lineáris programozási feladat aktuális adataitól, tehát a fordítás eredményét célszerű megőrizni. Ezért a REPFILE-nak (REPDATA-nak) permanens file-nak kell lennie.

A *SETREP* eljárás a lefordított programot és az éppen élő work matrixot összekapcsolja és a programot futáskész állapotban a REPWORK nevű file-on helyezi el.

Végül a *REPORT* eljárás a lefordított, összeszerkesztett programot a REPWORK-ről, az adatokat a REPDATA-ról beolvasva végrehajtja az előírt feladatokat.

4. Konkrét tapasztalatok

Több kis méretű és jelentőségű lineáris programozási feladat mellett egy nagyobb méretű modellt építettünk fel és vizsgáltunk a Bevezetésben említett két számítógépen.

A kisebb méretű feladatok, mint említettük, CMS alatt is futtathatók, ha ez installálva van. A CMS alatti futtatás a felhasználó számára nagyon kényelmes, hiszen ilyenkor rövid idő alatt számos vizsgálatot el lehet végezni, és az adatok kezelése nagyon egyszerű.

A nagy modell egy vállalat hosszútávú stratégiai tervezési feladata volt. A modell három ötéves tervidőszakot ölelt fel. Időszakonként 300—350 feltétel és 450—550 változó tartozott a feladathoz; így összesen mintegy ezer feltétel és ezerötszáz változó alkotta a mátrixot. Az egyes időszakok közötti összefüggés szűk keresztmetszeten jelentkezett, ezért a mátrix lényegében három diagonálisan elhelyezkedő almátrixból állt.

A stratégiai tervezést 12 fejlesztési körhöz kapcsolódóan 31 integer változó segítségével írtuk le. Egy-egy fejlesztési körhöz 2—4 integer változó tartozott, amelyek mindegyike csak a 0 vagy az 1 értéket vehette fel; az azonos fejlesztési körhöz tartozó változók összege kötelezően 1 volt. Ez azt jelenti, hogy a feladat megengedett megoldásában minden egyes fejlesztési körben pontosan egy integer változó értéke egyenlő 1-gyel, az összes többié 0-val. Az ilyen jellegű integer változó csoportokat a MIP/370 külön, ún. SOS-ként (*S*pecial *O*rdered *S*et) kezeli, az ilyen feladatok megoldásánál némileg más algoritmust használ és így könnyebben jut eredményre. Ezek az integer változók jelentették tehát a döntéseket. Minden fejlesztési kör a vállalat egy nagyobb részterületének felelt meg. Az egyes integer változók reprezentálták az erre a területre vonatkozó, egymástól eltérő fejlesztési pályákat, amelyek közül pontosan egynek kell megvalósulnia. Amennyiben egy integer változó értéke a megoldásban 1, úgy a hozzá kapcsolódó beruházások, kapacitások, egyéb ráfordítások (munkaerő, munkabér, karbantartás stb.) „élnek", az ehhez a fejlesztési körhöz tartozó többi változó által meghatározott adatok nem játszanak szerepet a megoldás többi részének meghatározásában. Az integer változók adatai természetesen mind a három tervidőszak tevékenységét meghatározták, így az ezekhez a változókhoz tartozó mátrix-elemek nem korlátozódtak egy almátrixra.

A három tulajdonképpeni tervidőszak mellett két fiktív ötéves időszakot is illesztettünk a modellhez. Ezek szerepe pusztán annyi volt, hogy biztosítsa a harmadik időszak végén elért szint fenntartását, valamint a pénzügyi egyensúly továbbvitelét. E két fiktív időszak mérete (kb. 20—20 változó és 15—15 feltétel) az egész modellhez képest elhanyagolható volt. Viszont mind az öt időszak pénzügyi elszámolásánál alkalmaztunk egy 0—1 értékű változót, amelynek a következő volt a szerepe: megengedte a vállalati tartalék megnyitását, ha az előző időszakban nem volt pénzmaradvány, ellenkező esetben viszont nem engedett a tartalékhoz nyúlni.

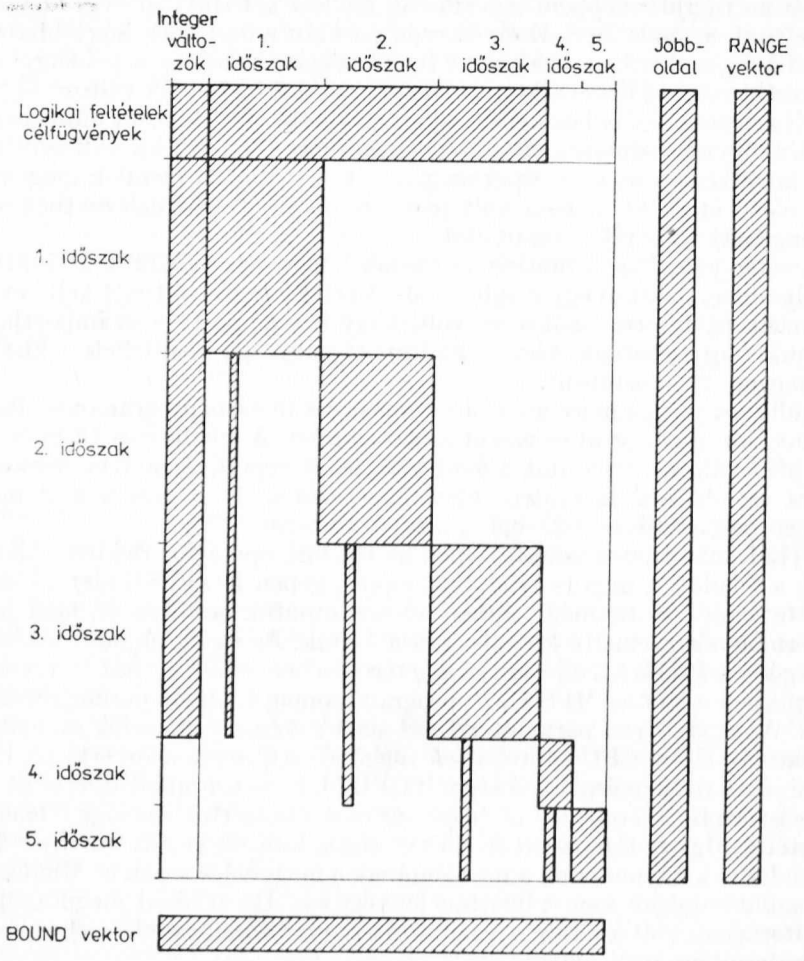
A hasonló jellegű nem-lineáris kapcsolatok leírására az MPSX-MIP/370 kiválóan alkalmas, pusztán egy megfelelő 0—1 értékű egész változót kell bevezetni. Egy másik példa erre: szükséges volt, hogy egy adott termék importját megengedjük, vagy letiltsuk; viszont az import nagysága nem lehetett kisebb egy előre megadott konstansnál.

Végülis egy eléggé nagy méretű vegyes egész lineáris programozási feladatot kaptunk, kb. ezer sorral és ezerötszáz oszloppal. A feladatban 12 SOS feltétel és 31 SOS változó, valamint 8 0—1 változó szerepelt. A mátrix szerkezetét a 2. ábra szemlélteti. A mátrix elemeinek száma 7—8 ezer között mozgott, sűrűsége elég stabilan 0,2—0,3% között változott.

Az IBM 3031 típusú számítógépen az OS/VS1 operációs rendszer alatt vizsgáltuk a feladatot, míg az IBM 4331 típusú gépen DOS/VSE alatt. Emiatt az egyes futások között, még teljesen azonos inputok esetében is, nem jelentős, de azért figyelemreméltó különbségek adódtak. Az egyik ok, ami felelős a különbségért, a PRIMAL eljárás paraméterezésében rejlik. A mátrix invertálásának szükségességét az MPSX/370 programcsomag többféle paraméterrel is irányítja. Az egyik ilyen paraméter DOS alatt a végzett iterációk számától, míg OS alatt az eltelt CPU időtől teszi függővé, szükséges-e invertálni. (A DOS ugyanis nem tudja mérni a felhasznált CPU időt, csak a futási időt, amit viszont nem célszerű figyelembe venni, hiszen ez csak kis mértékben függ a feladat természetétől). Így aztán az OS és a DOS alatti futások között bizonyos különbségek adódnak ugyanannak a problémának a megoldása során is. Mindazonáltal ezek a különbségek nem túlzottan lényegesek. Ha például megnézzük, hogy hány iterációra volt szükség a feasibilitás vagy optimalitás eléréséhez, az eltérés minden esetben 10% alatt volt.

All-slack állapotból indulva az első lépésben, a CRASH hatására, mintegy 700—800 iteráció után 100—150 helyen maradt infeasibilitás, amit további 500—600 iterációval a PRIMAL eljárással meg lehetett szüntetni, ha volt megengedett megoldás. A második fázisban a PRIMAL még 800—1000 iterációt végzett a folytonos optimum eléréséig, ha volt ilyen. Tehát az első megengedett megoldást kb. 1200, a folytonos optimumot kb. 2000 iterációval lehetett megkapni. A folytonos optimum elérésére az IBM 3031-es gép kb. 4 perc CPU időt fordított, míg ugyanez az IBM 4331-esnek kb. 15 perc CPU időbe került. (A futási idő nagysága nem jó mérőeszköz a gyorsaságra, hatékonyságra, hiszen az nagymértékben függ attól, milyen más programokkal van párhuzamosan leterhelve a számítógép.)

A folytonos optimum elérése után a MIXFLOW meglepően rövid idő alatt találta meg az optimális integer megoldást. Ez CPU időben mindig 2 perc alatt megtörtént és a megvizsgált node-ok száma sohasem haladta meg a 15-öt. Ebből is látszik, hogy a gyakorlatban nem kell félni nagyméretű fától. Érdekes tapasztalat az is, hogy a MIP/370, a mi feladatunk esetében, szinte minden eset-



2. ábra

ben olyan integer megoldást talált első ízben, ami rögtön optimális is volt. A futtatásoknak csupán pár százalékánál fordult elő az ellenkező eset, vagyis az, hogy az első integer megoldás még nem volt optimális. Háromnál több különböző integer megoldás azonban sohasem született: a mi esetünkben legkésőbb a harmadik integer megoldás optimális volt.

Komoly nehézségeink az MPSX-MIP/370 programcsomag belső tartalmából fakadóan csak egy ízben voltak. Ekkor a mixed fázisban nem kaptunk megengedett megoldást, de ezt semmi sem indokolta. Bizonyos toleranciák gondos megválasztásával végül sikerült a megfelelő eljárásokat pontosabb számolásra kényszeríteni, ami a megfelelő eredményre vezetett.

Feladataink megoldása során természetesen igen ritkán indítottunk all-slack állapotból. Általában az első ilyen futás eredményéből továbbszámolva lehetett előállítani az újabb és újabb megoldásokat. A feladat kismértékű megváltozta-

tása esetén nagyon hamar megengedett ill. optimális megoldáshoz lehetett jutni. De amikor nagyobb mértékű átalakításra volt szükség (az adatok kb. 10%-át kellett megváltoztatni), akkor is elég volt az eltárolt megoldás alapján pár száz új iterációt végezni az első megengedett megoldás eléréséig, és az optimalitásig is kb. ennyit.

Inkább a konkrét feladat belső sajátosságaiból származó tapasztalatunk, és az MPSX-MIP/370 programcsomag felépítésével csak kevéssé függ össze, hogy a kapott megoldás nagyon sok alternatív optimumot tartalmazott. Egy másodlagos célfüggvény bevezetésével, előírva, hogy az elsődleges célfüggvény értéke szinten maradjon, ezeknek az alternatív optimumoknak a túlnyomó része eltűnt. Az ECL nyelv használatával el lehet érni, hogy ilyen jellegű másodlagos célfüggvények vizsgálatára egyetlen job lefuttatása folyamán is sort kerítsünk.

A megoldások értékelésénél az MPSX/370 eljárásain kívül használtuk a Report Generatort is. A használat során felszínre került a Report Generator néhány hibája. Az egyik hibája a nehézkesség: különösen a feltételtől függő elágazások megszervezése bonyolult. Ezenkívül időnként kifejezetten zavaró az, hogy keveset „tud" (pl. tesztelni csak numerikus értékeket lehet). A nehézkesség folyományaként egy-egy program nagyon nagy terjedelmű, és emiatt a karbantartás is nehézkessé válik. Illusztrálásként csak annyit említünk meg, hogy modellünk input adatrendszere mintegy 5000 rekordból áll; a futás utáni report kb. 10 féle, képernyőnyi méretű táblázatához szintén kb. 5000 rekordból álló programot kell írni, ezenkívül a report futtatásához szükség van egy külön kb. 600 rekordból álló adatrendszerre is.

Összességében azonban az MPSX-MIP/370 programcsomagról igen jó benyomásokat szereztünk: viszonylag egyszerű feladatok esetében kezelése tulajdonképpen primitívnek mondható, mégis igen hatékony eljárásokkal rendelkezik; kevés igényre nem tud „felelni". A programcsomag tervezése olyan, hogy egyszerűbb feladatok esetében elegendő viszonylag felületesen megismerni; bonyolultabb esetekben már alaposan tanulmányozni kell, míg a különösen problematikus feladatoknál kifejezetten mélyen meg kell érteni a kínált lehetőségek közötti összefüggéseket. Így aztán mindenki megtalálhatja benne azt, amire éppen szüksége van.

(Beérkezett: 1984. november 13-án.)

IRODALOM

1. IBM Mathematical Programming System Extended/370 Program Reference Manual SH19 — 1095 — 2.
2. IBM Mathematical Programming System Extended/370 Mixed Integer Programming/370 Program reference Manual SH19 — 1091 — 1.
3. IBM Mathematical Programming System Extended/370 Control Languages SH19 — 1094 — 2.
4. IBM Mathematical Programming System Extended/370 Introduction to the Extended Control Language SH19 — 1147 — 0.
5. KÉRI G.: *Az operációkutatás számítógépes módszerei I.* Budapest, Tankönyvkiadó, 1984.